Ph.D. Dissertation

István Juhos

Ph.D. School in Computer Science
University of Szeged

Szeged 2009

Quotient and Power methods for the Graph Colouring Problem

Author: *István Juhos*
Ph.D. candidate
Department of Computer Algorithms and Artificial Intelligence

Advisor: *Prof. János Csirik*
Head of the
Department of Computer Algorithms and Artificial Intelligence

Szeged 2009

A thesis submitted for the degree of doctor of philosophy
of the University of Szeged

University of Szeged
Ph.D. School in Computer Science

Contents

Preface

History of graph colouring.

The first results about graph colouring deal almost exclusively with planar graphs in the form of map colouring. When trying to colour a map of the counties of England, Francis Guthrie postulated the four colour conjecture, noting that four colours were sufficient to colour the map, so that no regions sharing a common border got the same colour. Guthrie's brother passed on the question to his mathematics teacher Augustus de Morgan at University College London, who mentioned it in a letter to William Hamilton in 1852. Arthur Cayley raised the problem at a meeting of the London Mathematical Society in 1879. The same year, Alfred Kempe published a paper that claimed to have settled the question, and for a decade the four colour problem was considered solved. For his accomplishment Kempe was elected a fellow of the Royal Society and later President of the London Mathematical Society [106].

In 1890, Heawood pointed out that Kempe's argument was faulty. However, in that paper he proved the five colour theorem, saying that every planar map can be coloured with no more than five colours, using ideas of Kempe. In the following century, a vast amount of work and theories were developed to reduce the number of colours to four, until the four colour theorem was finally proved in 1976 by Kenneth Appel and Wolfgang Haken. Perhaps surprisingly, the proof went back to the ideas of Heawood and Kempe and largely disregarded the intervening developments [156]. The proof of the four colour theorem is also noteworthy for being the first major computer-aided proof.

In 1912, George David Birkhoff introduced the chromatic polynomial to study the colouring problems, which was generalised to the Tutte polynomial by Tutte, important structures in algebraic graph theory. Kempe had already drawn attention to the general, non-planar case in 1879 [90], and many results on generalisations of planar graph colouring to surfaces of higher order followed in the early 20th century.

Graph colouring has been studied as an algorithmic problem since the early 1970s. The chromatic number problem is one of Karp's 21 NP-complete problems from 1972, around the time of various exponential-time algorithms based on backtracking and heuristics. One of the major applications of graph colouring – register allocation in compilers – was introduced in 1981.

This thesis was motivated by Zykov's result in 1949, where he introduced his deletion–contraction recurrence theorem in [161; 162]. Though this theorem is well-

known in the literature, it has not received much attention in the algorithm design field until now. Zykov's approach makes a connection between different graphs through his edge deletion and vertex contraction operations. From a colouring point of view, these graphs may have the same properties. As Hell and Nešetřil describe in their work [85] in 2004, these operations can be expressed through graph homomorphisms. However, the homomorphism approach does not provide any implementation or any algorithm, but the approach motivates algorithmic steps. *The author* designed special homomorphism classes for the graph colouring problem with different implementations in ([96–101]). Despite the implementations being different, a general framework has been worked out to form a basis for a new colouring approach. This thesis is about *the author's* results and it contains various novel colouring strategies within a new framework.

István Juhos

Acknowledgements

First of all, I would like to thank my supervisor, Prof. János Csirik for supporting my work during my studies, providing calm and inspiring conditions for research, and for the useful and motivating discussions. I am deeply indebted to Jano van Hemert for the fruitful research collaboration and his constant and valuable support; and to Prof. Imre Bálint for the discussions and collaboration from which I learnt much, and who helped shape my ideas for the thesis. And many thanks to Attila Tóth, Dr. László Makra and Dr. György Szarvas for the fruitful research collaboration. I would also like to express my gratitude to David Curley, Betti Dobó and Csaba Dobó for their fine work in scrutinising and correcting this thesis from a linguistic point of view. The following people were also of great help in my studies: Prof. Gusztáv Eiben, Balázs Erdőhelyi, Dr. Tibor Gyimóthy, Dr. Péter Hajnal, Dr. András Hócza, Prof. László Lovász, Dr. Marc Schönauer, Dr. Michele Sebag, Phillip Tann, Dr. Masaru Tezuka, Dr. János Virágh and Jennifer Willies. Moreover, thanks to everyone who helped me to realise my goals, especially the High Performance Computing Group of the University of Szeged, the High Performance Computing Group of the University of the Hungarian Academy and the High Performance Computing Group of the University of Edinburgh, United Kingdom.

Special thanks to my family Virág, Lili and Léna, who gave me so much encouragement and motivation, and to my parents for their unconditional and loving support throughout my life.

Notation

$\mathbb{N}$	the set of all natural numbers
$\mathbb{R}$	the set of all real numbers
$\mathbf{x}$	column vector of real numbers
$\mathbf{x^T}$	transpose of vector $\mathbf{x}$, a row vector
$\|\mathbf{x}\|_i$	$p-$norm of vector $\mathbf{x}$: $(\sum_i \|\mathbf{x}_i\|^i)^{\frac{1}{i}}$
$\|\mathbf{x}\|$	Length of vector $\mathbf{x}$: $\|\mathbf{x}\| = \|\mathbf{x}\|_2$
$\mathbf{x} \parallel \mathbf{y}$	Vectors $\mathbf{x}$ and $\mathbf{y}$ are parallel: $\frac{\mathbf{x}^T\mathbf{y}}{\|x\|\|y\|} = 1$.
$\mathbf{x} \perp \mathbf{y}$	Vectors $\mathbf{x}$ and $\mathbf{y}$ are orthogonal: $\frac{\mathbf{x}^T\mathbf{y}}{\|x\|\|y\|} = 0$.
$\mathbf{x}_i$	$i-$th component of vector $\mathbf{x}$
X	a matrix of real numbers
$\|X\|_i$	Induced $i-$norm of matrix X: $\|X\|_i = \max\left\{\frac{\|X\mathbf{x}\|_i}{\|\mathbf{x}\|_i}\right\}$
$\|X\|$	Entrywise $2-$norm of matrix X: $\sqrt{\sum_{i,j} X_{ij}^2}$
X_i	$i-$-th row of matrix X
$X_{\_j}$	$j-$th column of matrix X.
X_{ij} or x_{ij}	(i,j) element of matrix X: $x_{ij} := X_{ij}$
$\mathbf{x} \vee \mathbf{y}$	element-wise OR operation between two binary $\{0,1\}^n$ vectors $\mathbf{x}$ and $\mathbf{y}$
$\mathbf{x} \cdot \mathbf{y}$ or $\langle \mathbf{x}, \mathbf{y} \rangle$	dot product of vectors $\mathbf{x}$ and $\mathbf{y}$: $\mathbf{x^T y}$
$\mathbf{x} \otimes \mathbf{y}$	dyadic product of vectors $\mathbf{x}$ and $\mathbf{y}$: $\mathbf{x y^T}$
$\mathrm{Diag}(\mathbf{x})$	Zero matrix with $\mathbf{x}$ in the main diagonal. Direct sum of the elements: $\bigoplus_i \mathbf{x}_i$
$\mathbf{e}$	vector of all ones.
I	identity matrix: $\mathrm{Diag}(\mathbf{e})$.
J	matrix with all its entries being one: $\mathbf{e} \otimes \mathbf{e}$.
J^{ij}	a matrix, where entry $J_{ij} = 1$, otherwise zero: $J^{ij} = I_i \otimes I_j$.
P^{ij}	a permutation matrix: $I - J^{ii} + J^{ij}$.
$X \vee Y$	element-wise OR operation between two binary $\{0,1\}^{n \times n}$ matrices X and Y
$X \bullet Y$ or $\langle X, Y \rangle$	sum of element-wise product of matrices X and Y (see dot product).
$X \circ Y$	element-wise product of matrices X and Y (Hadamard-Schur product).
$\mathrm{diag}(\mathbf{X})$	a vector formed by the main diagonal of the X matrix: $(X \circ I)\,\mathbf{e}$
G	a graph: $G = (V, E)$ or $G = (V_G, E_G)$
V	vertex set
E	edge set: $E \subseteq V \times V$
C	colour set
A_G	adjacency matrix of graph G (abbreviated form: A)
$(\cdot)^{col}$	select coloured objects, e.g. coloured vertices V^{col}
$(\cdot)^{unc}$	select uncoloured objects, e.g. uncoloured vertices V^{unc}

n	number of vertices: $\lvert V\rvert$
m	number of edges: $\lvert E\rvert$
k	number of colours: $\lvert C\rvert$
χ	chromatic number
ω	clique number
α	independence number
θ	Lovász-theta
π	permutation of {1,2,...,n} elements
$[x = y]$	The Kronecker delta function. $[x = x] = 1$ and if $y \neq x$, then $[x = y] = 0$.
$f(x) = \mathcal{O}(g(x))$	$\lvert f(x)\rvert \leq c \cdot \lvert g(x)\rvert$, where $c \geq 0$ for $x > x_0$, f and g are functions of x

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Contributions of the thesis

This thesis offers a general framework for graph colouring methods, where the traditional colouring scheme is defined via special graph homomorphisms motivated by [85; 161; 162]. These special homomorphisms proved useful in the design of algorithms by *the author* ([94; 96–102]). Three main reasons can be given for why this framework is useful.

First, this approach in general provides a potential decrease of the computational cost of colouring algorithms. In order to achieve this goal, special homomorphisms are applied which subsequently reduce the problem. In a parallel implementation, these reduction steps can be performed as one atomic operation, hence they do not introduce any extra computational effort. This helps algorithms to run faster.

Second, it provides a uniform and compact way in which algorithms can be defined. Embedding algorithms in the same common framework supports both their structural and performance comparison, which can be anyway problematic. Furthermore, it may give a deeper and comparable insight into the structure of algorithms. The framework itself generalises the formal colouring approach. With this generalisation an algorithm can be extended in a natural way, which may result in new algorithms.

Third, it opens the way to novel applications that extract useful information to help algorithms during their search. On one hand, a problem reduction step may reveal the skeleton of the problem and this may lead to a reconsideration of previous assumptions in a strategy. Hence existing algorithms can be enhanced after being embedded in the framework. On the other hand, the novel problem description results in novel information that can be used to extract and support a new scheme of the colouring process where new aspects can be identified.

This thesis has been organised so as to demonstrate and highlight these advantages via examples, experimental results and theoretical observations.

1.2 Overview of the Thesis

This thesis summarises the results obtained by *the author* over the past few years. The results can be separated into different groups according to the parts of the graph colouring framework developed by *the author*:

Concept The author defined the problem via certain graph homomorphisms. The author called these *Quotient and Power methods*.

Model The author described the concepts by concrete representations with suitable operations, resulting in his *Merge Models* with his nomenclature. Merge Models provide a novel description of the colouring problem. The operations, i.e. the *Merge Operations*, subsequently change the state of the model and direct it to a possible solution of the original graph colouring problem.

Strategy The author developed strategies in the model (*Merge Strategies*), which define possible directions toward a solution.

Algorithm The author constructed general frameworks (*Merge Frameworks*) in which strategies can be embedded. These frameworks in conjunction with the strategies form colouring algorithms (*Merge Algorithms*). Such algorithms generate a sequence of model operations according to the strategy to provide a candidate solution for the original problem.

Thesis points according to *the author*'s publications [94–102]

THESIS 1 *The author*, applying certain graph homomorphisms, defined two general concepts to redefine the graph colouring problem, namely the Quotient and Power methods [96; 99; 100]. He provided a concrete description of the general methods using matrix representations and Merge Operation of the rows or columns. He called these descriptions Merge Models. Based on the Merge Models the original problem undergoes an evolution and produces homomorphic graph images. These models can be a basis of novel and existing algorithms too. Embedding an algorithm into a Merge Model may considerably decrease its computational efforts. Moreover, such an embedding supports the structural analysis of the algorithms in a common way and makes available a natural extension of them, which may result in an increase in their performance. Traditional colouring schemes distinguish between the colours and the vertices of the graph. Merge Models integrate them into one single object. This anticipates a uniform algorithm design, where colour choices do not differ from the vertex choices.

THESIS 2 Based on the Merge Models of the colouring, *the author* unified and generalised the formal sequential colouring model in three different Merge Frameworks [100; 101]. These frameworks provide a uniform and compact description in which algorithms can be defined and analysed in the

same systematic way. Furthermore, exploiting the uniform description, he sketched some explanations of how the structure of algorithms can have an influence on the overall performance. Existing sequential colouring algorithms fit into one of the Merge Frameworks, and the frameworks provide novel approaches for algorithm design.

THESIS 3 *The author* provided a way to reduce the computational cost of colouring algorithms after embedding them into a Merge Framework [97; 99]. This improvement was demonstrated and analysed via experiments as well. In the experiments he analysed the phase transitions of different algorithms implemented in different Merge Frameworks. Furthermore, *the author* provided a natural extension of sequential colouring algorithms in the Merge Framework, which results in an increase in their efficiency.

THESIS 4 In each Merge Model the colouring operation is replaced by a Merge Operation. Several Merge Strategies were developed by *the author*. Since the models use matrix representations, he was able to define some of his strategies by applying special matrix row operations as well as matrix norms. The novel strategies of *the author* are listed below:

- Extended Hajnal; Extended Welsh-Powell (∞–norm) [97]
- Spectral norm[101]
- Spectral norm approximations [101]
- Dot product (entrywise norms) [97]
- Cosine [97]
- Zykov-tree and Lovász-theta [94; 102]

These strategies can be combined with different Merge Models and Merge Frameworks to form different algorithms. The performance analysis of these strategies are given. The novel algorithms are compared with several well-known benchmark algorithms. The novel algorithms outperformed the well-known algorithms in a standard benchmark set of graph instances. Moreover, their efficiency revealed in a more difficult-to-solve graph instance set, where the graphs are generated during the phase transition region, where finding a solution becomes really hard. In this case, the comparison is fair; that is, it cannot be manipulated by a good choice of the benchmark instances since the generated instances represent well all instances from difficult-to-solve graph classes.

THESIS 5 *The author* introduced the notion of a *Merge Path* in [101]. A Merge Path arises from the properties of the dynamically changing model during its evolution. Elements of such a path are associated with colouring steps. He was able to describe an abstract graph colouring approach based on Merge Paths, which allows the application of artificial intelligence methods in graph colouring e.g.:

– Using a training set of known graphs, a supervised learning algorithm [95] can learn certain optimal Merge Paths that are associated with optimal colouring steps. Then using the learnt knowledge, colouring steps for an unknown graph instance can be predicted.

– In an unsupervised learning task optimal Merge Paths of known graphs are clustered. Then unknown graphs, which are not involved in the clustering, can be classified in order to predict their properties such as their chromatic number.

THESIS 6 He embedded his colouring strategies into a meta heuristic, an evolutionary algorithm and created the following evolutionary operators for colouring [96–98; 101] :

– A mutation operator by acquiring difficult vertices in a candidate solution and forcing their early colouring

– A fitness function which solves the fitness granularity problem of the colouring

These novel meta heuristic algorithms performed well in an experimental comparison with different benchmark algorithms, on different benchmark graphs and difficult-to-solve generated problem sets as well.

Table 1.1 contains cross-references between thesis points and publications.

	[95]	[96]	[97]	[98]	[99]	[100]	[101]	[102]
THESIS-1		•			•	•		
THESIS-2						•	•	
THESIS-3			•		•			
THESIS-4			•				•	•
THESIS-5	•						•	
THESIS-6		•	•	•			•	

Table 1.1: Cross-reference between thesis points and publications

1.3 Overview of the chapters

This section provides an overview of how the publications of *the author* are related to the chapters of the thesis as well as to the thesis points.

Chapter 1 gives an overview of the thesis.

Chapter 2 summarises necessary definitions which will be used in this thesis.

Chapter 3 introduces the Graph Colouring Problem. It consists of definitions and analyses it from several aspects. It details important structural properties of graphs which may have an influence on the solution of the problem. Exploiting some structural features, we offer some simplification techniques of the original problem. Furthermore, we give an insight into the problem difficulty by complexity results and characterise hard-to-solve problem instances, which are a basis of our experimental investigations. In our analysis various bounds are provided to restrict the search space exploration. We overview the possible search spaces of the different representations of the problem.

Chapter 4 outlines the related work published in this field in the literature. It discusses some important real-life applications of graph colouring, providing graph instances from different sources. We describe various approaches available to solve the Graph Colouring Problem. Afterwards, we discuss several well-known graph colouring algorithms. The algorithms detailed with the provided graph instances serve as benchmarks in our experimental investigations.

Chapter 5 discusses graph homomorphism approaches of the Graph Colouring Problem and its consequences.

Chapter 6 introduces special graph homomorphisms for the colouring problem forming the Quotient and Power methods for the Graph Colouring Problem defined by *the author* in [96; 99; 100].

Chapter 7 describes the modelling of special graph homomorphisms by the so-called Merge Models using special matrix representations and matrix operations devised by *the author* in [96; 99; 100]. This chapter introduces different structures which may help colouring algorithms and which employ Merge Models.

Chapter 8 defines Merge Frameworks based on the Merge Models (see *Juhos et al.* [100; 102]). These frameworks are generalisations of the traditional sequential colouring schemes. They provide a general algorithm frame to assist the design and implementation of colouring algorithms. These frameworks also contain abstract strategies for the algorithm steps.

Chapter 9 introduces novel strategies for the algorithm steps which can be embedded into a Merge Framework to form an algorithm defined by *the author* in [95–98; 101; 102]. It consists of the analysis of the strategies besides their definition. The analysis shows a natural enhancement possibility of existing strategies when they are embedded into a Merge Framework. Example extensions are provided based on two well-known strategies. In another result of the analysis, a general idea for the strategy design is included, which offers a way for the application artificial intelligence methods in the colouring process.

Chapter 10 contains different novel Merge Algorithms introduced by *the author* in [96–98; 101; 102]. A Merge Framework with concrete Merge Strategies form Merge Algorithms (colouring algorithms). A definition of well-known benchmark algorithms in a suitable Merge Framework is provided as well. A thorough experimental investigation compares the benchmark algorithms with the novel Merge Algorithms on several standard benchmark problem sets.

Chapter 11 analyses the novel Merge Models and Merge Algorithms from various aspects, providing proofs for their efficiency. It includes efficient hardware and software implementation details as well [99; 100].

Chapter 12 This chapter consists of an appendix providing further interesting details about the Graph Colouring Problem and a summary of the thesis in English and in Hungarian.

Table 1.2 shows cross-references between the chapters and the publications.

	[95]	[96]	[97]	[98]	[99]	[100]	[101]	[102]	
Ch. 6		•			•	•			Quotient and Power methods
Ch. 7		•			•	•			Merge Models
Ch. 8						•		•	Merge Frameworks
Ch. 9	•	•	•	•			•	•	Merge Strategies
Ch.10		•	•	•			•	•	Merge Algorithms
Ch.11					•		•		Analysis

Table 1.2: Cross-reference between chapters and publications

Chapter 2

Preliminary definitions

This chapter summarises definitions which will be used in this thesis in accordance with [49; 138; 150]. Some general definitions have a slight restrictions for the sake of better utilisation in our topic.

Definition 2.1 (Graph) *A graph is a pair $G = (V, E)$ of disjoint finite sets, where $E \subseteq V \times V$. The elements of V are the vertices of the graph G, the elements of E are its edges.*

V_G denotes the vertex set and E_G denotes the edge set of the graph G, if the graph G must be emphasised in the notation. An edge between the vertices v and w is denoted by vw.

Definition 2.2 (Isomorphic) *$G = (V, E)$ and $G' = (V', E')$ are isomorphic graphs, if there is a bijection $\varphi : V \to V'$ with $vw \in E \Leftrightarrow \varphi(v)\varphi(w) \in E'$*

Definition 2.3 (Undirected and directed graph) *A graph G is said to be undirected, if the relation $E \subseteq V \times V$ is symmetric; otherwise, the graph is said to be directed.*

Unless it is explicitly stated, a graph is undirected. The edges of an undirected graph are called undirected edges and the edges of a directed graph are called directed edges.

Definition 2.4 (Loop edge) *The edge $e \in E$ of a graph G is a loop edge, if $e = vv$, where $v \in V$.*

Definition 2.5 (Incident) *A vertex v is incident with an edge e if $v \in e$; then e is an edge at v.*

The two vertices incident with an edge are its endvertices or ends, and an edge joins or connects its ends.

Definition 2.6 (Adjacent or neighbour vertices) *Two vertices v and w of G are adjacent, or neighbours, if $vw \in E$.*

Definition 2.7 (Edge density) *Let $G = (V, E)$ be a graph. The number $\frac{|E|}{\binom{n}{2}}$ is the edge density of G.*

Note that the value of $\binom{n}{2}$ is the maximum number of edges in a graph.

Definition 2.8 (Empty graph) *The empty graph is either the graph with no vertices and hence no edges or any graph with no edges.*

Definition 2.9 (Complete graph) *G is a complete graph if all its vertices are pairwise adjacent. A complete graph on n vertices will be denoted by K_n.*

Definition 2.10 (Regular graph) *A regular graph is a graph where each vertex has the same number of neighbours.*

Definition 2.11 (Simple graph) *A simple graph G is an undirected graph, which has no loop edge.*

An example of a simple graph can be seen in Figure 2.1.

$$V = \{v_1, v_2, v_3, v_4, v_5, v_6\}$$

$$E = \{v_1v_2, v_2v_3, v_3v_4, v_4v_5, v_5v_6, v_6v_1, v_3v_1, v_3v_6\}$$

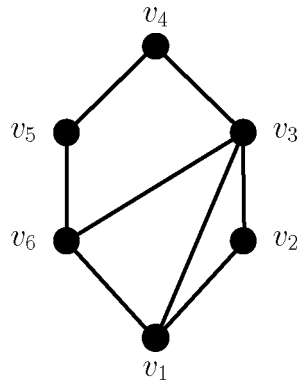


Figure 2.1: A simple graph

Definition 2.12 (Multigraph) *A multigraph is a pair $(V, \mathcal{E})$, which contains a vertex set V and an edge multiset $\mathcal{E}$. Where $\mathcal{E}$ consist of edges between any two vertices and multiple edges are permitted.*

If $V' \subseteq V$, then $G - V'$ is obtained from G by deleting all the vertices in $V' \cap V$ and their incident edges. If $V' = \{v\}$ is a singleton, $G - v$ is written rather than $G - \{v\}$. For $E' \subseteq V \times V$, $G - E' = (V, E \setminus E')$ and $G + E' = (V, E \cup E')$; furthermore for $e \in E$, $G - \{e\}$ and $G + \{e\}$ are abbreviated to $G - e$ and $G + e$.

Definition 2.13 (Complement graph) *$\bar{G}$ is the complement graph of graph G, if $V_{\bar{G}} = V_G$ and $E_{\bar{G}} = V_G \times V_G \setminus E_G$.*

Definition 2.14 (Sub-graph) *$G'(V', E')$ is a sub-graph of $G = (V, E)$, if $V' \subseteq V$ and $E' \subseteq E$. Denote it by $G' \subseteq G$.*

For example $H = (V', E')$ is a sub-graph of Figure 2.1, if $V' = \{v_1, v_2, v_3\}$ and $E' = \{\{v_1, v_2\}, \{v_2, v_3\}, \{v_3, v_1\}\}$. The graph H is a complete graph on three vertices, namely the K_3.

Definition 2.15 (Clique) *A **clique** is a complete sub-graph of a graph.*

Since H is a complete sub-graph of G, therefore it is a clique in G.

Definition 2.16 (Maximal clique) *A maximal clique is a complete sub-graph that is not contained in any other complete sub-graph*

Definition 2.17 (Maximum clique) *A maximum clique is a clique containing the largest possible number of vertices.*

A maximum clique is necessarily maximal, but the converse does not hold. Take v_4 and v_5 vertices of Figure 2.1 with the edge between them. They form a complete graph on two vertices, called K_2. This K_2 is not part of a larger clique in G, hence it is a maximal clique, but not a maximum because graph H, which is a K_3 is larger clique. H is the largest clique, and hence it is a maximal clique. Although, H is not unique. There may be more than one maximum and consequently several maximal cliques in a graph. Vertices v_1, v_3, v_6 with edges between them also form a K_3 clique, which is maximal too.

Definition 2.18 (Independent set) *An independent set of a graph is a subset of vertices such that no two of them are mutually adjacent.*

There is a strong connection between cliques and independent sets since an independent set of a graph is a clique in the complement graph.

Definition 2.19 (Maximal independent set) *A maximal independent set is an independent set that is not a subset of any other independent set.*

A graph may have different maximal independent sets of widely varying sizes as we saw in the case of cliques.

Definition 2.20 (Maximum independent set) *A maximum independent set is an independent set containing the largest possible number of vertices.*

A vertex which is not in a maximum independent set must be connected to a member of the set. Otherwise, the vertex in question should be member of the maximal independent set. Take an example vertex set $S = \{v_1, v_3, v_5\}$ from our example graph in Figure 2.1. It is a maximum independent set and the vertices which are not included $\{v_2, v_4, v_6\}$ are adjacent to one of the vertex in this set and form another maximal independent set.

Definition 2.21 (Neighbour set) *Let G be a graph and $v \in V$. Neighbour set is the set of neighbour vertices of v and denoted by $N(v)$: $N(v) = \{w : vw \in E\}$.*

$N(.)$ can be extended to set of vertices. If S is a set of vertices, then $N(S) = \bigcup_{v \in S} N(v)$. If S is a maximum independent set of graph G, then $V \setminus S = N(S)$ Otherwise, if there was a vertex $u \notin N(S)$ then S would not be maximum, since $S \cup u$ would be a larger independent set.

Definition 2.22 (Vertex degree) *The degree of a vertex v is the number of its neighbours: $d(v) = |N(v)|$.*

The minimal vertex degree in a graph is denoted by δ, while the maximum is denoted by Δ. From now on an we will use an abbreviated form of $d(v_i)$, denoted by d_i, where $v_i \in V$.

Definition 2.23 (Dominating Set) *D is a dominating set of vertices of a graph G, if $D \subset V_G$ and $N(D) = V_G \setminus D$.*

A dominating set covers all vertices of a graph which are not included. Dominating sets are closely related to independent sets. An independent set is also a dominating set if and only if it is a maximal independent set. Hence, any maximal independent set in a graph is necessarily a minimal dominating set as well.

Definition 2.24 (Dominated and dominant vertex) *v is a dominated vertex by a set of vertices $D \subset V_G$ of a graph G, if $N(v) = N(v) \cap N(D)$. A dominated vertex has neighbours which are all adjacent to some other vertex, the dominant vertex.*

Definition 2.25 (Partition) *A set $\{V_1, \ldots, V_k\}$ of disjoint subsets [1] of a set V is a partition of V if $V = \bigcup_{i=1}^{k} V_i$ and $V_i \neq \emptyset$ for every i.*

Definition 2.26 (Vertex contraction) *Vertex contraction is an operation where two vertices are replaced by one single vertex. If $u, v \in V$, then $G/\{u, v\}$ or G/uv represents the graph after contraction of the u, v vertices.*

A vertex contraction can result in multiple edges when the contracted vertices were connected to the same vertex. Multiple edges created by a vertex contraction can be either kept or collapsed into one single edge. Keeping or collapsing will be marked, if it is not clear from the context.

Definition 2.27 (Edge contraction) *Edge contraction is the vertex contraction of two adjacent vertices. If $e \in E$, then G/e is the appropriate graph after the edge merge.*

From here on vertex contraction will mean contracting *unconnected* vertices only, otherwise we will use the term edge contraction. All these definitions will appear in some form in the thesis, but for our topic, vertex contraction will be the most important one that will crop up many times throughout this thesis.

[1] $\forall\ i, j\ V_i \cap V_j = \emptyset,\ i \neq j$

Chapter 3

Graph Colouring Problem

The graph colouring problem (GCP) is an important subset of constraint satisfaction problems [35; 62; 104; 145]. It has many real-world applications such as scheduling, register allocation in compilers, frequency assignment and pattern matching [1; 18; 25–27; 31; 36; 47; 61; 108; 129; 130; 135]. Here the problem will now be defined as follows:

Definition 3.1 (Graph vertex k-colouring) *Let G be a graph and C a set of colours, where $|C| = k$. Graph k-colouring is a map of vertices to colours:*

$$c : V \xrightarrow{sur} C \quad , \quad v_i \mapsto c(v_i)$$

Put briefly, *graph vertex k-colouring* (or simply *graph k-colouring*) is an assignment of colours for each vertex. The problem occurs in the colouring process when we consider edges as constraints.

Definition 3.2 (Proper graph vertex k-colouring) *A proper graph vertex k-colouring, if it exists, is a k-colouring where adjacent vertices are assigned different colours:*

$$c : V \xrightarrow{sur} C \quad , \quad v_i \mapsto c(v_i) \quad , \quad \forall (v_i, v_j) \in E \Rightarrow c(v_i) \neq c(v_j)$$

Definition 3.3 (Graph minimum vertex colouring) *Graph minimum vertex χ-colouring is a proper χ-colouring where χ is the smallest integer needed to get a proper colouring.*

The smallest number of colours that can properly colour vertices is called the **chromatic number** of a graph and will be denoted by χ. A graph G is k-**colourable** if its vertices can be coloured properly by k colours; in other words, if its chromatic number; is at most k. It will be called k-**chromatic** if k is its chromatic number. In a particular colouring, a subset of vertices assigned to the same colour is called a **colour class**. Figure 3.1 shows a proper colouring, which is minimum as well. Sets $\{v_1, v_5\}, \{v_2, v_4, v_6\}$ and $\{v_3\}$ are the colour classes in Figure 3.1. Definition of k-colouring consists of a condition for the existence of such an assignment. The parameter k plays very important role on the feasibility of a k-colouring. Section 3.4 provides several bounds on the feasibility. Without any qualification, the colouring of a graph is always a proper vertex colouring so that no two adjacent vertices receive the same colour as seen in

Figure 3.1. Although, Section 3.1.1 describes other type of colourings as well. Since a vertex with a loop edge could never be coloured properly, it is understood that graphs in this context are loopless. Moreover, it is reasonable to restrict the colouring to simple graphs, where the edges are undirected. However, different graph types can be seen in several generalisations of the problem. The terminology of using colours goes back to map colouring initiated and analysed by the following authors in the period of 1852-1890: Guthrie and De Morgan and Hamilton and Heawood and Cayley and Kempe and Heawood [4; 8; 65; 83; 84; 157]. Colours like red and blue are only used when the number of colours is small, but generally colour names are substituted by numbers $1, 2, 3, \ldots$, however they will be referred as colours. Unless stated otherwise, the unqualified term '*graph*' usually refers to a simple graph. Without loss of generality we shall only consider connected graphs[1], that is when there is only one component. Figure 3.1 shows a proper colouring. Colours are shown as different angle coloured half circles in certain figures where colours are used for the sake of clarity.

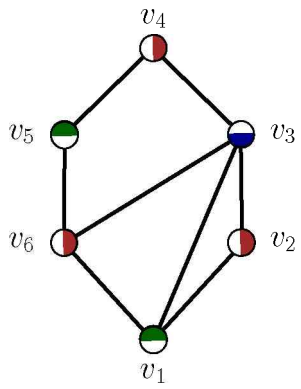


Figure 3.1: A proper colouring of a graph.

A colouring using at most k colours is called a k-colouring. There are three main questions:

1. Can a graph be coloured with k colours?
2. What is a k-colouring of a graph, if it exists?
3. How many k-colourings exist for a graph?

This thesis concentrates on the algorithmic aspects of the colouring, hence we will mostly be concerned with answering question 2, but we cannot ignore the other two questions stated above. The two other questions can assist us in algorithm design. However, providing a $k-$colouring (question 2.) serves to answer the $k-$colourability (question 1), but sometimes the $k-$colourability itself can be answered more easily, without providing any colouring. It is often important to know whether a graph is $k-$colourable or not, e.g. before starting an expensive $k-$colouring algorithm. Several methods have been developed to quickly determine a $k-$colourability. These methods usually provide bounds for k, and some of these bounds can be found in Section 3.4. A trivial example, when the answer to question 1 is straightforward, is when k is greater

[1]There is a path from any vertex to any other vertex in the graph.

than the maximum vertex degree. In this case k colours are quite sufficient for a proper colouring as each vertex has fewer neighbours than the available number of colours. Non-trivial examples are the perfect graphs where the chromatic number is computable in polynomial time according to [105; 110; 111]. Usually, when task is the $k-$colouring, finding one solution for a colour assignment is enough. Nevertheless, Section 3.1.4 shows how important is to analyse the whole set of solutions before we design an efficient colouring algorithm. This chapter collects necessary and interesting information about graph colouring to motivate and help understanding the rest of the thesis. To have better insight into the problem, it provides problem analysis besides the introduction of definitions and concepts.

3.1 Graph colouring definitions

3.1.1 Improper colouring and semicolouring

We saw an example of a proper colouring in Figure 3.1. Now we shall focus on proper colourings, but first we shall mention other colouring types as well because they can be also useful in the design of proper colouring algorithms. One possibility is when an algorithm tries to exploit the features where not only proper but improper colourings are also available.

Definition 3.4 (Graph vertex improper colouring) *An improper colouring is a colouring where at least one constraint is violated:*

$$c : V \xrightarrow{sur} C \quad , \quad v_i \mapsto c(v_i) \quad , \quad \exists (v_i, v_j) \in E \Rightarrow c(v_i) = c(v_j)$$

Solvers which apply improper colouring have to cope with the violated colouring, so they have to correct the colouring to get a proper colouring. On the one hand, this relaxation of the problem, where improper colourings are available leads to broader search space. This has its drawbacks since it is more likely that we will not find a solution. On the other hand, it removes the constraints temporarily in the space exploration which supports more flexible algorithm design. We can design search paths through those elements of the space which are not available in proper colouring. These paths can provide a shortcut to a solution.

Definition 3.5 (Graph vertex k-semicolouring [103]) *A graph k-colouring is a k-semicolouring if at least half of the vertices are coloured properly.*

$$c : V \xrightarrow{sur} C, v_i \mapsto c(v_i), \quad \forall (v_i, v_j) \in E' \Rightarrow c(v_i) \neq c(v_j), E' \subseteq E, |E|/2 \leq |E'|$$

Relying on a semicolouring, one can design an $\mathcal{O}(k(n) \log_2 n)$-colouring algorithms according to the following [103].

Lemma 3.1.1 *If an algorithm can $k(n)$-semicolour any n-vertex graph G, where $k(n)$ increases with n, then it could be used to $\mathcal{O}(k(n) \log_2 n)$-colour G.*

Recall our previous statements, G is a simple graph and k-colouring means proper k-colouring. Moreover, contracting two vertices refers to unconnected vertices and edge contraction will identify contraction of connected vertices. Multiple edges created by a vertex contraction can be either kept or collapsed into one single edge. Keeping or collapsing will be marked if it is not clear from the context.

3.1.2 Chromatic and Achromatic number

Minimum colouring can be defined as finding a partition of the vertex set into minimum number of independent sets. Consequently, the union of such two independent sets results in a non-independent set, otherwise we would be able to reduce the number of components in the partition.

$$\chi = \min\{k : \{V_i\}_{i=1}^{k} \text{ partition of } V,\ V_i \cap E = \emptyset\,,\ (V_i \cup V_j) \cap E \neq \emptyset\} \tag{3.3}$$

The maximisation of this expression (Eq. 3.3) leads to another important number namely the achromatic number.

$$\psi = \max\{k : \{V_i\}_{i=1}^{k} \text{ partition of } V,\ V_i \cap E = \emptyset\,,\ (V_i \cup V_j) \cap E \neq \emptyset\}$$

The achromatic number tells us how badly a colouring algorithm can perform. The number of colours used by and algorithm is between these two numbers, but they rarely attain these bounds (see Section 3.3). There are two additional numbers which can have a big influence on the performance of an algorithm, namely the clique and the independence number.

3.1.3 Clique and independence number

Definition 3.6 (Clique Number) *The clique number $\omega(G)$ of a graph G is the number of vertices in a maximum clique of G.*

The problem of computing the clique number for a given graph is an NP-complete problem ([82; 139]). Since a clique is a complete sub-graph, a complete graph requires as many colours as the number of its vertices for a proper colouring. Hence at least as many colours are needed for a proper colouring of a graph as the size of its maximum clique. This holds $\omega \leq \chi$. According to Motzkin and Straus, formulation [66; 127] cliques can be characterised by a submatrix in the adjacency matrix (Figure 3.2(a) and 3.3). The submatrix of an adjacency matrix A which belongs to a clique is a matrix with every entry equal to one except the main diagonal, which has zeros [2] (Figure 3.2(a)). To mask out this submatrix one can use a characteristic matrix of the edges of the clique (Figure 3.2(b)) which has ones in the appropriate positions, otherwise it contains zeros. The problem of clique finding turns into the problem of finding an appropriate clique mask which masks out a clique submatrix from the adjacency matrix.

[2] The 0-s have been replaced by dots for the sake of clarity.

	v_1	v_2	v_3	v_4	v_5	v_6
v_1	**0**	**1**	**1**	·	·	1
v_2	**1**	**0**	**1**	·	·	·
v_3	**1**	**1**	**0**	1	·	1
v_4	·	·	1	·	1	·
v_5	·	·	·	1	·	1
v_6	1	·	1	·	1	·

(a) A clique submatrix in the adjacency matrix

	x_1	x_2	x_3	x_4	x_5	x_6
x_1	1	1	1	·	·	·
x_2	1	1	1	·	·	·
x_3	1	1	1	·	·	·
x_4	·	·	·	·	·	·
x_5	·	·	·	·	·	·
x_6	·	·	·	·	·	·

(b) Characteristic matrix of the clique.

Figure 3.2: Matrices belongs to clique of $\{v_1, v_2, v_3\}$ vertices.

Definition 3.7 (Independence number) *The independence number $\alpha(G)$ of a graph G is the cardinality of the largest independent set of G.*

Finding a maximum independent set in essence means finding of a maximum clique in the complementer graph. Consequently, the problem tight is the same, i.e. NP-complete. As k-colouring defines a partition of V into k-independent sets $\{V_i\}_{i=1}^{k}$, the following holds $\chi \leq \frac{|V|}{\alpha(G)}$. Colour classes are independent sets in a minimum colouring. One can think of finding suitable independent sets which form appropriate partition of the vertex set. Therefore it is reasonable to examine the number of independent sets in a graph. The number of independent sets also represent the number of maximal cliques due to their complementary nature. The number of *different* size maximal independent sets as well as the number of maximal cliques is between $n - \log n - \mathcal{O}(\log \log n)$ and $n - \log n$ according to Erdős [54]. A similar formulation can be given for independent sets as for cliques using the adjacency matrix of the complementer graph. The submatrix of an adjacency matrix which corresponds to an independent set is a matrix with all entries equal to zero, i.e. the opposite of the clique submatrix case (see Figure 9.8). Ordering the rows and relevant columns of an adjacency matrix according to colour classes in a k-colouring, we get the k number of zero blocks in the main diagonal. Hence, a colouring problem can be formulated by these zero blocks, as described in Section 3.1.4.

3.1.4 Colouring matrices

According to Section 3.1.3, the submatrix of the adjacency matrix which corresponds to an independent set is an all zero matrix. Colour classes form independent sets with the associated zero submatrices. The entries of these zero submatrices define the corresponding colouring, where the vertices belongs to an all zero submatrix get the same colour. These relations can be expressed by a colouring matrix $X = (x_{ij})$, which is a $\{0, 1\}$ matrix. It is defined by the conditions

$$x_{ij} = \begin{cases} 1 & \text{if } c(v_i) = c(v_j) \\ 0 & \text{otherwise} \end{cases} \tag{3.5}$$

An extreme example is the identity matrix, which colours each vertex differently. In

contrast, the matrix of all ones assigns just one colour for each vertex. With a complete graph needs identity colouring matrix for proper colouring, the colouring matrix with all one entries is suitable for the empty graph only. Consider the $\{V_i\}_{i=1}^{k}$ partition of vertices into k independent sets, which provides a k-colouring, where V_i is the i-th colour class of n_i number of vertices. Taking vertices in the order of the colour classes, we can describe the colouring by a colouring matrix which has 1 blocks in the main diagonal and zeros elsewhere. A few examples can be found in Figure 9.9. We note that such a matrix has the following properties ([51; 124]): $(v_i, v_j) \in E \Rightarrow x_{ij} = 0$; X is symmetric; X is positive semi-definite: $X \succeq 0$.

3.2 Number of colourings

For a graph there may be several possible k-colourings. The following section is concerned with their cardinality. The number of colours in a k-colouring can be expressed by a polynomial, called the chromatic polynomial. The chromatic polynomial is defined as the unique polynomial of degree n through the points $(k, p(k))$ for $k = 0, 1, \ldots, n$.

Definition 3.8 (Chromatic polynomial) *The chromatic polynomial counts the number of ways a graph can be coloured using no more than a given number of colours. If the number of colours is k then the chromatic polynomial is denoted by $p(k)$.*

The values of the polynomial count the equivalent colourings as well (see an example in Section 11.7). The chromatic polynomial contains as much information about the colourability of G as the chromatic number does. Indeed, the chromatic number is the smallest positive integer that is not a root of the chromatic polynomial. Thus $\chi = \min\{k : p(k) > 0\}$. Section 11.8 contains an example for the chromatic polynomial.

3.3 Complexity

Unfortunately, there is no known convenient method for determining the chromatic number of an arbitrary graph. Determining whether a graph admits $k-$colouring is difficult in general. However there is a polynomial time algorithm for cases $k = 1$ and $k = 2$, but for $k \geq 3$ the problem becomes NP-complete [80; 104; 138]. Thus finding the chromatic number is, computationally, a hard problem. It is not only $NP-$complete, but there is also no polynominal time algorithm that can colour every graph G using fewer than $n^{\epsilon}\chi$ colours for a specific small positive constant ϵ [160]. To understand the difficulties involved better, the authors of [73; 144] demonstrated that colouring $3-$colourable graph with 4 colours is still $NP-$hard. As for the achromatic number, determining it is $NP-$hard as well (see [62]).

3.4 Bounds of the chromatic number

In Section 3.3 we saw how difficult is to approximate the chromatic number in general [160], but there are polynomial time computable bounds of the chromatic number which approximate well the chromatic number in particular cases. Moreover, the bounds allow us to relate the chromatic number and the structural properties of a graph. This section will describe such bounds.

Brooks' theorem states a relationship between the degree of a graph and its chromatic number. According to the theorem, for a graph where every vertex has at most Δ neighbours, the vertices may be coloured with just Δ colours, except for two cases. Complete graphs and cycle graphs of odd length require $\Delta + 1$ colours.

Brooks' [20]. If the graph is not complete or not an odd cycle, it has the following bound

$$\chi \leq \Delta \tag{3.6}$$

In Section 3.1.3 we gave bounds where the chromatic number is characterised by the size of the maximum clique and independence set.

Clique number and independence number.

$$\omega \leq \chi \leq \frac{n}{\alpha} \tag{3.7}$$

The difference between the ω and χ can be arbitrarily large [70; 128]. However, there is no effective way to determine the clique number, because it requires much effort as that for the chromatic number. But there is a graph property which can be efficiently computed. This property provides a better lower bound for the chromatic number than the clique number. Nevertheless, the difference between this property and the chromatic number can still be large [56]. Lovász introduced a graph property called $\bar{\theta}$ which is computable in polynomial time and gives a better lower bound than ω. $\bar{\theta}(G)$ is computed in the complementer graph $\bar{G}$, which explains the 'bar' symbol. In fact, it serves as the vector chromatic number, which is the solution of a relaxed graph colouring problem (see Section 9.9).

Lovász [109].

$$\omega \leq \bar{\theta} \leq \chi \tag{3.8}$$

The result is tight for perfect graphs where $\omega = \bar{\theta} = \chi$ Not only is the value of the property important, but the way it obtained can be very useful too (see Section 9.9). A study of $\bar{\theta}$ provides helpful information about graphs. It also has applications in other areas besides graph colouring.

Hoffman and Wilf [88; 154].

$$1 + \frac{\lambda_{max}}{-\lambda_{min}} \leq \chi \leq \lambda_{max} + 1 \tag{3.9}$$

The difference between the λ_{max} and χ can be large (see Section 3.7), but in Section 9.5 we shall see that this bound is useful in colouring algorithm design. Also, a relationship

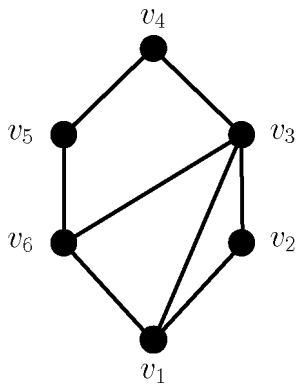
may be seen when we examine the relations between induced matrix norms. The spectral norm is the smallest among the induced norms. For graphs the spectral norm is equal to the largest eigenvalue λ_{max}. Since $\Delta = ||A||_1 = ||A||_\infty$ and $\lambda_{max} \leq \Delta$, hence

$$\chi \leq \lambda_{max} + 1 \leq \Delta + 1 \tag{3.10}$$

For regular graphs the equality $\lambda_{max} = \Delta$ holds. Both lower and upper bounds can help in algorithm design. Identifying the target colouring and the starting k can be an important in a colouring process. This can be achieved by determining an upper bound value. Knowing the lower bound may also be useful in the preparatory step of the colouring process, where a graph can be simplified by applying this bound. Vertices with degree lower than a lower bound value can be removed (see [29]), while an upper bound can be a target colour in the beginning of a minimal colouring process.

3.5 Characteristic polynomial

Many bound estimates in Section 3.4 were obtained from an algebraic analysis of the colouring problem. Usually, the analysis is based on some matrix which characterises the problem. Different matrices may be associated with a graph. One obvious example is the adjacency matrix, which is illustrated in Figure 3.3. This matrix encodes the



	v_1	v_2	v_3	v_4	v_5	v_6
v_1	·	1	1	·	·	1
v_2	1	·	1	·	·	·
v_3	1	1	·	1	·	1
v_4	·	·	1	·	1	·
v_5	·	·	·	1	·	1
v_6	1	·	1	·	1	·

Figure 3.3: A graph G and its adjacency matrix.

graph encoding important properties. Most notably its eigenvalues, its determinant and its trace. The eigenvalue problem for this is

$$A\mathbf{v} = \lambda\mathbf{v} \qquad \rightarrow \qquad (\lambda I - A)\mathbf{v} = 0 \tag{3.11}$$

Let the eigenvalues of this equation be λ_i, where $\lambda_{max} = \lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n = \lambda_{min}$, and let $\mathbf{v_1}, \mathbf{v_2}, \ldots \mathbf{v_n}$ be the eigenvectors, respectively. This is a well studied equation in the literature and there are several solvers which can compute all the solutions efficiently. Owing to the definition of the trace and the relation with the sum of eigenvalues, the following condition holds $\operatorname{tr} A = \sum_i a_{ii} = \sum_i \lambda_i = 0$. For non-bipartite graphs $\lambda_{max} > -\lambda_{min}$ [45], while for bipartite graphs the absolute values of the eigenvalues are equal. In Section 3.4 we showed how the eigenvalues can be used to set bounds on the chromatic number. Also, in sections 9.4.2 and 9.5 we describe

efficient colouring strategies based on the the principal (the largest) eigenvalue and the principal eigenvector. The eigenvalues as roots of a polynomial $p_A(x) = \Pi_i(x - \lambda_i)$ give the characteristic polynomial of A.

Definition 3.9 (Characteristic polynomial) *The characteristic polynomial of a matrix is defined by*

$$p_A(x) = \det(A - xI) \tag{3.12}$$

The characteristic polynomial of a graph is the characteristic polynomial of its adjacency matrix. It is a graph invariant, i.e. isomorphic graphs have the same characteristic polynomial, hence it is more interesting in the light of the great symmetry of the problem (see an example in Section 11.7). Next, write p_A in the form

$$p_a(x) = \det(xI - A) = x^n - c_1x^{n+1} + \cdots + (-1)^n c_n \tag{3.13}$$

The polynomial coefficients also encode interesting features of the graph which can be of help in the colouring process as well. Section 9.7 describes an application for colouring. From [9], the coefficients of the characteristic polynomial of a graph satisfy the conditions: $c_1 = 0$, $-c_2 = |E|$, $\frac{-c_3}{2}$, $(-1)^n c_n = \Pi_i \lambda_i = \det A$. Section 11.8 contains an example for the characteristic polynomial.

3.6 Search spaces

This section describes search spaces which generally arise in graph colouring algorithm design.

3.6.1 Permutation space

Using a greedy colour assignment, there is always a permutation of a vertices which generates a solution. Section 4.2 contains more details about the greedy colour assignment problem. Next, let us see a solution (Figure 3.1) and order vertices according to its colour class identifier, where the same colour vertices appear in a natural order (Figure 3.4). Note that the colour class identifiers may be changed without changing a solution. The vertices belonging to a colour class can be listed in not only a natural, but arbitrary order. Hence several permutations can result in the same performance colouring, producing a symmetry in the space. After creating a permutation of vertices, we do a greedy colour assignment to the vertices, in the order of their appearance in the permutation. Greedy colouring produces a colouring which requires no more colours than the original solution. This procedure may lead to different colourings. This is because a low degree vertex which does not have any neighbour in any colour class can get different colours, resulting in another optimal colouring. However they can be removed before starting a colouring (see Section 3.4). Based on this approach, an algorithm must search in the permutation space of the vertices. Even though the size of this search space is large ($n!$), it has been proved a better representation for sequential

colouring schemes in [7; 52] than the vectorial colour assignment scheme, which has k^n elements (see Section 3.6.3). The graph colouring problem has large symmetries (see an example in Section 11.7), thus the $n!$ space will be reduced considerably. Recall that the vertices belonging to the same colour class can appear in an arbitrary order. Hence, several permutations may result in the same candidate solution. An algorithm, which searches in the permutation space, always targets the minimum colouring case.

$$\pi = (v_1, v_5, v_2, v_4, v_6, v_3) \qquad c_g = (1, 1, 2, 2, 2, 3)$$

Figure 3.4: A solution represented as a permutation π in a greedy colouring c_g.

3.6.2 Independent set space

Since the colour classes form independent sets, a search can be performed in this space. The search itself may be a search of an appropriate characteristic vector which selects a suitable subset of the set of all independent sets. Although it assumes the generation of all independent sets, which is time and space consuming task as the number of independents sets can be huge (see Section 3.1.3). Therefore, a dynamic generation approach can be applied instead of the static one, which generates all sets in advance. Hence a search must be done in a dynamically changing environment, starting with e.g. the one-size independent sets, namely, the vertices. An algorithm can sequentially combine independent sets by performing a union of some of them. The interpretation of this approach in a colouring language might be a re-colouring scheme where the starting colour palette is n, which colours each vertices differently. In order to reduce the number of colours used, an algorithm must properly re-colour the vertices by using an $(n-1)$-colour palette. This process is continued until no further reduction is possible. Usually, a greedy variation of this approach is applied in the literature [70; 72; 77; 103; 152]. Namely, a colour is chosen in advance and some vertex choice strategy is applied. The colour chosen is assigned to each possible vertex. When no further such assignment is possible, an algorithm gets another colour and goes on in the same fashion for the remaining vertices, not affected by the previous colourings. In an independent set formulation, it means that one can find an independent set, that is a dominating set and where no further vertex can be encompassed. Then removing the dominating set, the algorithm continues this same strategy for the remaining vertices. Figure 3.5 shows the colour classes (independent sets) of the colouring of Figure 3.1.

$$S_1 = \{v_1, v_5\} \qquad S_2 = \{v_2, v_4, v_6\} \qquad S_3 = \{v_3\}$$

Figure 3.5: A solution represented as a set of independent sets.

3.6.3 Vector space

In Section 3.6.2 we introduced a search space where independent sets, i.e. colour classes were generated as constituents of the colouring. These approaches require a $\{0, 1\}$ characteristic vector $\mathbf{x}$ with k non-zero component which designates k independent sets, if the goal is a $k-$colouring. See a colouring in Figure 3.6 and its representation as

a characteristic vector in Figure 3.6. In the case of minimum colouring, the space of all possible $\{0,1\}$ vectors, whose dimension is the number of possible independent sets (see in Section 3.1.3). Although this approach is quite time-consuming. Unfortunately, it assumes the generation of all independent sets in a graph in advance, which requires 2^n independent set examinations, using a brute force approach. Furthermore a sophisticated generation of the independent sets may require extra computation effort. The benefit of this formulation is that it is possible for an algebraic method to be applied on a relaxed version of the problem, where $\mathbf{x}$ is no longer binary $\{0,1\}$ valued but real valued. The characteristic vector $\mathbf{x}$ can be decomposed into k characteristic vectors $\{\mathbf{v_i}\}_{i=1}^{k}$ that form a polyhedron. Each of these vectors have only one non-zero element, which designates an independent set. Hence, each v_i represents only one independent set. A colour assignment to a vertex may be simply interpreted as a discrete colour

$$\begin{array}{ccccc} \{S_1 & S_2 & S_3 & S_4 & \dots\} \\ (\,1 & 1 & 1 & 0 & \dots) \end{array}$$

Figure 3.6: Characteristics vector of independent sets S_1, S_2, S_3 of a solution in the set of all independent sets.

assignment function. The assignment can be represented by an n-tuple, an integer vector on n elements. The size of the search space is k^n in the case of $k-$colouring and n^n if we desire minimum colouring. However n^n can be reduced based on the upper bounds and lower bounds of the problem (cf. the bounds given in Section 3.4).

3.7 Random G_{n,p_e} graphs

There are many classes of graphs that could and should be used to test colouring algorithms. The most natural class is perhaps the class G_{n,p_e}, the random graphs, where n is the number of vertices, and for each pair of vertices an edge is assigned with probability p_e, i.e. an edge probability. This class of graphs has been extensively studied from a colouring aspect, especially for $p_e = \frac{1}{2}$, where the number of the possible instances is the biggest. According to [12; 13; 58; 71], asymptotically, for a fixed probability p_e and $b = \frac{1}{1-p_e}$, the chromatic number is almost surely be

$$\chi \sim \frac{n}{2\log_b n} \tag{3.14}$$

Furthermore, if $d = \frac{\sum_i d_i}{n}$ is the average degree in the graph, then the following holds $\chi \sim \frac{d}{2\ln d}$. The average degree in a graph is a lower bound for the largest eigenvalue [45; 46] [3] and hence the gap between the largest eigenvalue and the chromatic number can be arbitrary large.

[3] $d = \frac{\sum_i d_i}{n} = \frac{\mathbf{1}^T A \mathbf{1}}{\mathbf{1}^T \mathbf{1}} \leq \max_{\mathbf{x}} \frac{\mathbf{x}^T A \mathbf{x}}{\mathbf{x}^T \mathbf{x}} = \lambda_{max}$

3.8 Phase transition

Graph $k-$colouring exhibits a phase transition depending on the ratio of constraints to the maximum number of possible constraints $\binom{n}{k}$, where the number of solvable problem instances quickly drops to zero. At that transition constraint solvers require the most search effort to find solutions for solvable problem instances. Recent investigations [43] have shown a good explanation towards explaining the rise in difficulty during the phase transition. To demonstrate the existence of the phase transition, take a class of $k-$colourable random graphs, where the graph structure is known. Let $\mathcal{G}_{eq,n=200,p_e,k=5}$ be the set of $5-$colourable equipartite random graphs on 200 vertices. In the case of equipartite graphs, each colour class has nearly the same number of vertices. Moreover, p_e defines the edge probability (see random graphs in Section 3.7); whose value describes the number of the edges in the graph. Section 4.1 provides further details about equipartite graphs. A sequence of graphs are generated by modifying the edge probabilities from 0 to 1 in a systematic way. Hence the number of edges of the generated graphs is varied in a region called the phase transition. This is where hard-to-solve problem instances are generally found, which is shown using the typical easy-hard-easy pattern in Figure 3.7. The graphs are all equipartite, which means that in a solution each colour is used approximately as much as any other. The demonstration graphs are generated using a well-known graph $k-$colouring generator of Culberson [44]. The graph set consists of groups according to the following edge probabilities $p_e \in \{0.01, 0.03, 0.05, \ldots, 0.98\}$. Each group has ten graph instances generated by using the same p_e, but different random seeds $\{1, 2, \ldots, 10\}$ in the generation. The same random order of the vertices is fixed for each graph. Then a greedy colour assignment is applied for each graph; that is, taking the ordered vertices of a graph, the first vertex get the first colour and all following vertices get the first available colour which produced proper colouring. Section 4.2.2 contains more information about the greedy colouring and its analysis. This procedure is repeated nine times to get ten different random orders. Hence, there were ten colourings for each graph instances. The number of colours of the colourings was averaged for each p_e groups, i.e. number of colours of ten colourings for ten graphs of a p_e group resulted in 200 values. These averages with 95% confidence intervals are plotted in Figure 3.7. A competition graph set in the literature may contain only particular instances of graph families. Therefore, generated instances, which include the phase transition region, are necessary in an experimental comparison of different colouring algorithms.

3.9 Summary

In this chapter we provided an insight into the graph vertex $k-$colouring problem and investigated it from several angles. In the next chapter we will give an overview of the work done on this problem in the literature.

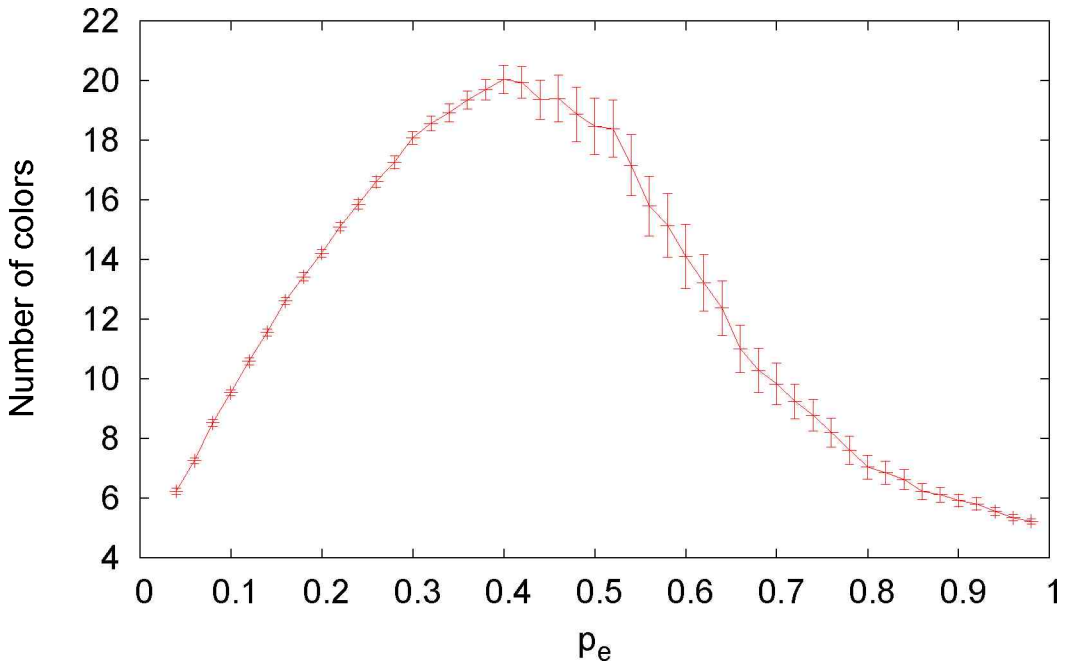


Figure 3.7: Phase transition of the greedy colouring in the $\mathcal{G}_{eq,n=200,p_e,k=5}$ graph class.

Chapter 4

Related work

Graph colouring has a wide range of real-world applications to solve constraint satisfaction problems, hence its results are of great intrest for researchers. Since the problem first appeared in the literature, a lot of research has been done in this field. This chapter reviews the major application areas of graph colouring and describes approaches for solving the problem. In addition, interesting graph instances and concrete algorithms are given with which the various approaches can be compared experimentally.

4.1 Benchmark graphs

This section describes a set of benchmark graph instances used in the experiments performed in Chapter 10. They are drawn from a large number of sources. Their description reveals how a real-world problem can be defined as a graph colouring problem.

The DIMACS archive. A common repository was created for testing colouring algorithms in the Second DIMACS Challenge [93] in 1992. Graph colouring was one of the problems addressed in the challenge. The purpose of the competition was to encourage and coordinate research in the experimental analysis of algorithms. Later the repository was extended by Trick adding other instances [143].

Random k-equipartite graphs. Next, define the following class of graphs to provide another testbed for our algorithms. Let $G_{eq,n,p_e,k}$ be the set of such G_{n,p_e} random graphs (see Section 3.7), where the vertex set of G is partitioned into k as nearly equal sized sets. These graphs form the class of random equipartite graphs. They are one of the most difficult to solve instances because they have the largest symmetry and the largest number of instances in the G_{n,p_e} class. To get an instance in this class one can $k-$partite vertices and draw edges only between the members of different subsets with a certain probability, i.e. an edge probability p_e. Of course, the chromatic number of such an instance is not greater than k, but it strongly depends on the edge probability. Indeed, for a zero edge probability the chromatic number drops to one. The edge probability of a $k-$partite graph lies in the inequality range $0 \leq p_e \leq \frac{(k-1)}{(n-1)} \cdot \frac{n}{k}$. Varying p systematically, experiments may reveal those regions where algorithms have difficulties, namely the phase transition region. In order to produce test instances of equipartite graphs, Culberson's graph generator [44] was used here.

4.2 Benchmark algorithms

This section provides benchmark algorithms which are commonly used to compare the performance of methods [17; 48; 70; 75; 151]. The performance of these methods serves as reference in the experimental comparison of the algorithms described in this thesis. Finding a solution of the colouring problem is hard in general, as shown in Section 3.3. Hence one can rely on heuristics which do not provide an exact solution, but only an approximate one. However, several algorithms have been developed to solve the graph vertex colouring problem, but none of them is optimal, their performance is always depending on the investigated problem. There are several polynomial time algorithms (see [70; 75; 77; 103; 151; 152]) that provide a guarantee for the approximate solution of the colouring for a given number of colours. Some of our benchmark algorithms have this guarantee, but for certain problems their performance can be worse than those algorithms which do not ensure such a guarantee.

4.2.1 Traditional sequential colouring schemes

According to the authors of [70], there are two traditional sequential colouring approaches. The first is the sequential colour assignments, where vertices get colours in a greedy manner; that is, each vertex gets the earliest available colour. Algorithms in this scheme take uncoloured vertices in order via some strategy then apply a sequential greedy colour assignment to the vertices, as described in Section 4.2.2. The search space is the space of vertex permutations, as seen in Section 3.6. This greedy colouring approach works well with several vertex ordering strategies, as shown in [17; 48; 75; 151]. These algorithms use some heuristics and usually without any guarantee of the number of colours to be used. Furthermore, there is always an order of the vertices which results in an optimal colouring with the greedy colouring approach. See Section 4.2.2 for further details. The second approach is the maximal independent set procedure, where instead of uncoloured vertices, the colours or colour classes are taken step by step. Colour classes form independent sets. At the start, there are no colour classes, hence it starts with an empty colour class, i.e. an independent set. Then it fills this set with uncoloured vertices, according to a vertex choosing strategy, until its saturated; that is, no more uncoloured vertices can be encompassed. Saturation occurs when the set becomes a dominating set; that is, each external vertex is connected to one of the internal vertices. In terms of a colour assignment, take the first colour and colour as many vertices as possible with the same colour. If a colour class is saturated, then it creates a new one and continues in the same fashion, That is, when no more uncoloured vertices can be coloured with the current colour, take a new colour and colour with the new colour as many uncoloured vertices as possible, repeating this step until uncoloured vertex exists. Algorithms which provide a guarantee for a maximum number of colours used in their colouring apply this approach [70; 77; 103; 151; 152]. This above approach can be interpreted as a *maximal independent set strategy* [70]. In fact, a colour choice is a choice of a colour class, which is an independent set. With this strategy, an empty independent set – a new colour – is created and filled with

vertices until it becomes maximal, i.e. when no more vertex addition is possible. To illustrate this, start with a single vertex, a one-element independent set and put as many non-neighbour vertices into this set as possible in order to increase the size of the independent set to its maximum. After a while the set becomes a maximal independent set, i.e. each non-included vertex will have a neighbour vertex in the independent set. Then continue with the rest of the vertices in the same fashion. In traditional sequential colouring schemes one is concerned with coloured and uncoloured sub-graphs, denoted by G^{col} and G^{col} respectively. Algorithms exploit information taken from both sub-graphs. Hence we shall now define some of their important properties:

Definition 4.1 (Uncoloured degree) *The uncoloured degree of a vertex v is the number of uncoloured neighbours of v:* $d^{unc}(v) = |\{v_j \,|\, v_j \in N(v)\ ,\ v_j \in G^{unc}\}|$.

Definition 4.2 (Coloured degree) *The coloured degree of a vertex v is the number of coloured neighbours of v:* $d^{col}(v) = |\{v_j \,|\, v_j \in N(v)\ ,\ v_j \in G^{col}\}|$

Definition 4.3 (Colour saturation degree) *The coloured saturation degree of a vertex v is the number of different neighbour colours of v:* $d^{sat}(v) = |\{c(v_j) \,|\, v_j \in N(v)\ ,\ v_j \in G^{col}\}|$

The same notation is used for their maximum and minimum as for the maximum degree Δ and minimum degree δ, with the inclusion of the appropriate superscripts $^{unc}, ^{col}, ^{sat}$.

4.2.2 Greedy colouring scheme $(\Delta + 1)$

Greedy colouring takes an order of the vertices and assigns colours sequentially to them in a greedy manner. That is, a vertex gets the earliest available colour. Vector $\mathbf{x}$ contains the sort keys of the ordering of the vertices. It is predetermined by a vertex ordering strategy.

GREEDY COLOURING ALGORITHM$(G, C, \mathbf{x})$
1 **for** $t \leftarrow 1$ **to** n
2 **do**
3 $\quad \mathbf{v} \leftarrow [\arg\max_{v_i}\{\mathbf{x}_i \,|\, v_i \in V^{unc}\}]$
4 $\quad v = \mathbf{v}_1$
5 $\quad c \leftarrow \min C \setminus \{c(v_i) \,|\, v_i \in N(v)\}$
6 $\quad c(v) \leftarrow c$
7 **return** $[\{c(v_i)\}_{i=1}^{n}]$

The greedy colouring method takes the uncoloured vertex v which has the maximum value in the sort key vector $\mathbf{x}$. Then it finds the earliest available colour c, then assigns c to v. The first available colour is that colour which has a minimum index and is not assigned to any of the neighbours of v. Note that $C = \{1, \ldots, k\}$ and every colour assignment implies that the $G^{col} = G^{col} + v$ and $G^{unc} = G^{unc} - v$ sub-graphs are updated each time. There may be the same degree vertices and they form a vector of currently

chosen vertices $\mathbf{v}$ in the order of their choice. The algorithm always chooses the first $\mathbf{v}_1$ among them. The greedy colouring procedure does not provide any strategy for ordering the vertices. Many heuristics exploit the power of the greedy performance and try to further refine the upper bound of the number of colours used. These heuristics explore the space of the vertex permutations. The vertex choice of these heuristics can result in a set of vertices if no other choice is present, the first being chosen among them by taking a natural order. A vertex ordering heuristics which uses greedy colouring cannot perform worse than $\Delta + 1$ as they keep the colouring below the Brook's bound (see Section 3.4). Nevertheless, for particular graphs the greedy performance varies greatly, but for large G_{n,p_e} random graphs, almost surely, it consumes approximately $\frac{n}{\log_b n}$ colours, where $b = \frac{1}{1-p_e}$ ([70; 71]); that is, approximately twice as many as the chromatic number (see Section 3.7). Section 3.8 detailed an experiment where greedy colouring was performed by taking $5-$colourable equipartite random graphs on 200 vertices. The phase transition occurred when the edge probability approached $p_e = 0.4$; that is the performance of the greedy colouring became worse in this region. The expected number of colours is $\frac{n}{\log_b n} = \frac{200}{\log_{1.\hat{6}} 200} = 19.28$, which seems quite good after analysing the plot of Figure 3.7. However, for other p_e-s, the expression gives over and underestimations: for $p_e = 0.2$ it is 8.42 and for $p_e = 0.8$ it gives 60.75. Note that the expression belongs to random graphs and our analysis covers only random equipartite graphs on 200 vertices. Nevertheless, this expression characterises well the greedy colouring at the peak of the phase transition.

4.2.3 Welsh-Powell $(\max_i \min\{d_i + 1, i\})$

The Welsh-Powell heuristic approach [151] is based on the greedy algorithm. The basis of the uncoloured vertex choice is the degree. This variant of the greedy colouring applies a vertex ordering. Vertices are ordered according to decreasing vertex degrees. The $d_i = d(v_i)$ is the degree of the vertex in the i-th position in the ordering. Then greedy colouring is applied to the vertices in order, which uses at most $\max_i \min\{d_i + 1, i\}$. Let $[\ .\ ]$ be an operation which generates a vector from the elements of a set, taking a natural order.

WELSH-POWELL COLOURING ALGORITHM(G, C)
1 **for** $t \leftarrow 1$ **to** n
2 **do**
3 $\quad \mathbf{v} \leftarrow [\arg\max_{v_i \in V^{unc}} d_G(v_i)]$
4 $\quad v = \mathbf{v}_1$
5 $\quad c \leftarrow \min C \setminus \{c(v_i) \,|\, v_i \in N(v)\}$
6 $\quad c(v) \leftarrow c$
7 **return** $[\{c(v_i)\}_{i=1}^{n}]$

The colouring constraints are specified by the edges. The Welsh-Powell heuristic considers the highest degree vertex as the most constrained one. i.e the most difficult one to colour. However, during a sequential colouring the colouring constraints are specified by the number of neighbouring colours. It is a variant of the greedy colouring where the $\mathbf{x} = \mathbf{d}$, where $\mathbf{d}$ contains the degrees of the vertices.

4.2.4 Hajnal $(\lambda_{max} + 1)$

The Hajnal heuristic approach [75; 153] applies a similar assumption as the Welsh-Powell heuristic approach, but its bound may be better than $\Delta + 1$. The maximum number of colours used by this heuristic is equal to the maximum [1], i.e. the principal (the largest) eigenvalue λ_{max} of the adjacency matrix of G. This bound is provided by a greedy colour assignment where the order of the vertices is determined by the components of the principal eigenvector [2]. Each eigenvector component is associated with a vertex according to the corresponding adjacency matrix rows/columns. This variant of the greedy colouring defines the sort key vector $\mathbf{x}$ by the components of the principal eigenvector.

HAJNAL COLOURING ALGORITHM$(G, C, \mathbf{x})$
1 **for** $t \leftarrow 1$ **to** n
2 **do**
3 $\quad \mathbf{v} \leftarrow [\arg\max_{v_i}\{\mathbf{x}_i \mid v_i \in V^{unc}\}]$
4 $\quad v = \mathbf{v}_1$
5 $\quad c \leftarrow \min C \setminus \{c(v_i) \mid v_i \in N(v)\}$
6 $\quad c(v) \leftarrow c$
7 **return** $[\{c(v_i)\}_{i=1}^{n}]$

4.2.5 DSatur of Brèlaz

The DSatur heuristic [17] rely on the colour or colour saturation degree d^{sat} of vertices in the current state of the colouring, i.e the number of different neighbour colours of a vertex. The Welsh-Powell and the Hajnal heuristic do not consider the state of the current colouring, but DSatur does it. Hence it is reasonable to distinguish between the coloured and uncoloured sub-graphs of the original graph G. Objects belonging to the coloured or uncoloured sub-graphs are denoted by ^{col} and ^{unc} subscripts, respectively, e.g. V^{col} and $V^{unc} = V \setminus V^{col}$. DSatur chooses the most constrained vertex in terms of the colour degree; that is, it chooses the maximum colour degree vertex and performs a greedy colouring on it. Since DSatur does not re-colour, there is no sense in using the colour degree for the already coloured vertices. For tie breaking, when more than one vertex has the same colour degree, the Welsh-Powell heuristic is applied in the G^{unc} graph. It looks for the vertex that has the highest uncoloured degree Δ^{unc} among the uncoloured vertices. The uncoloured degree $d^{unc} = d_{G^{unc}}$ is then calculated in the uncoloured graph G^{unc}, i.e. the edges of coloured vertices are not taken into account.

[1] For graphs $\lambda_{max} \geq -\lambda_{min}$, equality occurs only in the case of bipartite-graphs.
[2] The adjacency matrix is symmetric, hence the right and the left eigenvectors are the same.

DSatur colouring algorithm(G, C)

```
1  for i ← 1 to n
2  do
3      U ← {v_i | d^sat(v_i) = Δ^sat, v_i ∈ V^unc}
4      v ← [v_i | d^unc(v_i) = Δ^unc, v_i ∈ U]
5      v = v_1
6      c = 1 + min C \ {c(v_j) | v_j ∈ N(v)}
7      c(v) = c
8  return [{c(v_i)}_{i=1}^n]
```

Dsatur put the most saturated vertices into the U set. If there are more than one such vertex, then applies a trial for tie breaking by the degrees of the elements of U. The final tie breaking is performed by choosing the first of the same maximum degree vertices of U as seen in the greedy colouring scheme. Then a the vertex chosen gets a colour by a greedy colour assignment.

4.2.6 Erdős $(\mathcal{O}(n \log n))$

The Erdős heuristic makes similar assumptions as DSatur but in the opposite way, however he recommended it for a theoretical analysis, and several algorithms apply his principle or similar assumptions [77; 152]. An Erdős $O(n/\log n)$ heuristic [70, p. 245] works as follows. First, take the first colour and assign it to the vertex v that has the minimum degree. Vertex v and its neighbours are removed from the graph. Continue this in the remaining sub-graph in the same fashion until the sub-graph becomes empty, then take the next colour and use the algorithm for the non-coloured vertices and so on until each vertex is assigned a colour. This approach guarantees $\mathcal{O}(n/\log \chi(n))$ number of colours in the worst case. However, an algorithm which has proved bounds for the number of colours used in a colouring makes an exact analysis possible, but other algorithms without such a bound can perform better in many cases. Next, separate the coloured and uncoloured sub-graphs as well, as described in Section 4.2.5 for the DSatur heuristics. Here the minimum degree of the uncoloured vertices will be denoted by $\delta^{unc}(v) = \delta_{G^{unc}}(v) = \min_i\{d^{unc}(v_i) \mid v_i \in V^{unc}\}$.
Let V_c be the colour class of c, i.e. the set of the same coloured vertices.

Erdős colouring algorithm(G, C)

```
1  c ← 1
2  for i ← 1 to n
3  do
4      v ← [v_i | d^unc(v_i) = δ^unc, N(v_i) ∩ V_c = ∅, v_i ∈ V^unc]
5      if v = [ ]
6        then c ← c + 1
7        else  v ← v_1
8              c(v) = c // v becomes the member of V_c
9  return [{c(v_i)}_{i=1}^n]
```

The vector $\mathbf{v}$ consists of the uncoloured minimum degree vertices, which can get the current colour c, keeping the rule of the proper colouring; that is, the neighbours of

these vertices cannot be in the colour class V_c. The first vertex of these vertices is selected and c is assigned to it. The last colour c is updated only when $\mathbf{v}$ is empty, i.e. there is no further vertex which can get a colour c.

4.2.7 Evolutionary algorithm – standard fitness

An evolutionary algorithm (EA) is a subset of evolutionary computation, a generic population-based metaheuristic optimisation algorithm. One popular approach for dealing with graph $k-$colouring is evolutionary computation [40; 42; 52; 55; 68; 133], where a set of candidate solutions (the population) is continuously changed (evolved) until it fulfils a certain stop condition. The evolution of the population is divided into generations. Candidate solutions can be modified or combined, creating new candidates between two generations. We use a standard steady state evolutionary algorithm [6; 53] to search through the space of permutations (see Section 3.6.1). The steady state model keeps the size of the population constant throughout the generations. This algorithm maintains a population Π of permutations of the vertices. Each permutation π is evaluated by the so-called fitness function $f(\pi)$, which defines the goodness of a candidate solution π. Here, $f(\pi) = k(\pi) - \hat{\chi}$, where $\hat{\chi}$ is a lower bound of the chromatic number (e.g. 1). The $k(\pi)$ determines the number of colours used by a greedy colouring (see Section 4.2.2), using the π order of the vertices. Randomly generated permutations form the initial population. Then the appropriate fitness values are calculated in each generation. After doing the fitness calculation each candidate solution is modified (mutated) by a certain probability p_{mut} and each candidate pair is combined (recombinated) based on another probability p_{xover} to get new candidate solutions in the search space. Recombination or crossover is the common name of the two operand change operators, which produce one or two new permutation(s). A selection is performed in the set of the original and new elements of the population to create the next population. This procedure continues until the stop condition is satisfied. Then the greedy colouring by the best candidate solution, i.e. permutation, provides the output of the algorithm. The settings of the evolutionary algorithm :

INITIALISATION: uniform random generation of permutations.

MUTATION: simple swap mutation, which selects at random two different elements in the permutation and then swaps them (see Figure 4.1(b)).

CROSSOVER: $2-$point order based crossover (ox2), as shown in Figure 4.1(a). The two permutations π_1 and π_2 are cut at two points. The first and the last part of the permutations are inserted without any change into the two new candidate solutions π_1' and π_2'. After the central part in the new permutations is ordered according to the element order in the other permutations.

SELECTION: $2-$tournament selection, where it employs elitism of size one; that is, it keeps the best candidate solution. Tournament selection involves running several "tournaments" among a 2 individuals chosen at random from the population. The winner of each tournament (the one with the best fitness) is selected.

STOP CONDITION: the algorithm terminates, when it reaches a certain number of generations or number of fitness evaluations. Furthermore, when an optimal

solution is found, here, usually the fitness is zero at an optimum point. In our case, it is achievable if $\hat{\chi} = \chi$.

$EA^{swap,ox2}_{k-\hat{\chi}}(G, C)$

```
1  Π ← random permutations(population size)
2  while termination condition
3  do
4      for π ∈ Π    // Evaluate each permutation
5      do
6          k(π) ← max{Greedy colouring(G, C, π)}    //Number of colours used
7          f(π) ← (k_π − χ̂)    // Fitness [a]
8      Π = Π ∪ mutation(Π, p_mut) ∪ crossover(Π, p_xover)
9      Π = selection(Π, f)
10 π ← best(Π, f)
11 return Greedy colouring(G, C, π)
```

[a] $\hat{\chi}$ is a lower bound of χ.

π_1	=	v_6	$\mathring{v}_1$	$\mathring{v}_4$	v_3	v_2	v_5
π_2	=	v_3	v_2	v_5	$\mathring{v}_4$	v_6	$\mathring{v}_1$
π'_1	=	v_6	$\mathring{v}_4$	$\mathring{v}_1$	v_3	v_2	v_5
π'_2	=	v_3	v_2	v_5	v_4	v_6	v_1

(a) Order-based crossover (ox2).

$$v_1\ v_4\ v_3\ v_2\ v_5\ v_6 \xrightarrow{swap(v_4,v_5)} v_1\ v_5\ v_3\ v_2\ v_4\ v_6$$

(b) Swap mutation

Figure 4.1: EA operators. Elements v_1, v_4 of π_1 are ordered according to the order of these elements in π_2 in Fig.4.1(a)

4.2.8 Evolutionary algorithm – Stepwise adaptation of weights

The Stepwise Adaptation of Weights (SAW) was introduced in [52] as a very promising technique for colouring graph 3-colouring problems. The basic idea behind SAW is to learn on-line about the difficulty of constraints in a problem instance. This is achieved by keeping a vector of weights that associates the weights with constraints. In the context of graph k-colouring, every edge is assigned a weight. These weights then get initial values of one. Next, a basic evolutionary algorithm is used to solve a given problem instance. Every generation is interrupted in order to vary the vector of weights using the best individual of the current population. Every constraint violated by this individual is incremented by one. Then the evolutionary algorithm uses this new vector. The fitness of an individual equals the sum of the weights of all the constraints it violates. By adapting this fitness function using the vector of weights may prevent the evolutionary algorithm from getting stuck in local optima.

4.3 Algorithm approaches

Extending the list of Section 4.2, this section discusses other frequently used approaches as well to solve the graph colouring problem. Lots of algorithms have been created and studied to solve the graph minimum vertex colouring problem. Actually, these algorithms come in two main types: the exact algorithms where finding of a solution is guaranteed, but the time involved may be considerable due to the complexity of the problem (see Section 3.3); and the non-exact, the approximation algorithms where however a solution is not guaranteed but one may find a solution or a good approximation of it in a reasonable time. The latter methods may have stochastic components. Some recent surveys of these methods can be found in [60; 91; 114; 158] The graph colouring problem can be exactly solved by an exhaustive search, i.e. systematically exploring a search space [43; 44; 93]. Unfortunately, when the size of the instances grows the running time for exhaustive search soon become prohibitively large, even for instances of fairly small size. To improve the efficiency of the search, several heuristics were developed to generate a 'good' starting candidate solution which may be close to an optimal solution [17; 48; 70; 75; 77; 108; 116; 131; 137; 151–153]. Then starting the exploration process with the generated candidate solution, a systematic search can considerably improve the performance. Usually, the exploration is based on an examination of the local environment of the generated solution and it assumes that a neighbourhood relation is defined on the elements of the search space. This approach led to the development of local search methods [5; 24; 30; 60; 79; 87]. These methods usually apply some heuristic to generate a new candidate solution from an existing one in its local environment. But though a heuristic can considerably improve a solution they do not always provide an optimal solution, hence these methods belongs to the class of approximate algorithms. Many algorithms studied today employ a stochastic process in the local search to guide a candidate solution to a suboptimal solution or, hopefully, to an optimal solution. Several of these approaches maintain a population of candidate solutions. Examples of such methods include tabu-search [10; 87], simulated annealing [28; 92] and ant colony optimisation [21; 39]. One popular approach for dealing with graph colouring is evolutionary computation [6; 40; 42; 52; 55; 59; 68; 81; 115; 133; 145]. In the development of algorithms for graph colouring, various integer programming formulations of the problem could be used. Several such formulations, usually involving binary variables, have been proposed. These variables can identify different structures: e.g. independent sets [118]; a variable for each possible colour and vertex [33; 120; 122]; acyclic orientations of a graph [57]. In several formulations an optimal solution can be represented as a binary vector of the variables. These binary vectors constitute a polytope, a colouring polytope. These polytopes are the central topics of the problem analysis [22; 67]. Several relaxed versions of these integer programmes have been developed to approximate a face of a colouring polytope [50; 103; 118; 121; 136]. Different techniques may improve the efficiency of these methods e.g. column generation with branch-and-bound [23; 118; 136] or branch-and-cut [122]. Actually the branch-and-bound technique implicitly uses Zykov's idea (see [136]). This idea is detailed separately in the next section.

4.3.1 Zykov-tree approach

In the middle of the last century Zykov came with the idea, of applying an edge addition or vertex contraction instead of a colour assignment in the colouring problem. During these operations new graphs are created from the original one which may inherit the parent graph's properties.

Theorem 4.1 *(Zykov theorem [162]) Let G be a graph. If $\{v, w\} \notin E$, then*

$$\chi(G) = \min\{\chi(G + vw), \chi(G/vw)\}$$

Proof

Let $\mathcal{C}$ be the set of proper colourings and $|c|$ the number of colours used by a colouring $c \in \mathcal{C}$, then $\chi = \min\{|c_i \in \mathcal{C}|\} = \min\{\min\{|c_i| : c_i(v) \neq c_i(w)\}, \min\{|c_i| : c_i(v) = c_i(w)\}\} = \min\{\chi(G + vw), \chi(G/vw)\}$ □

Two vertices v and w get either the same or different colour in any colouring. Therefore, there may be a contraction or edge between them. A Zykov binary tree ([14; 162]) is built on these two operations of two unconnected vertices of a graph. Here we connect them or contract them, keeping their neighbours with simple edges. According to the Zykov theorem, one of the result graphs with these operations has the same chromatic number as the original graph (see 4.2). The construction of the Zykov-tree is terminates, when no further reduction is possible. Hence, the leaves consist of complete graphs K_i. Each of them describes a colouring where contracted vertices get the same colours. Consequently, $\chi = \min_i |V(K_i)|$. The Zykov-tree is not uniquely determined, however. It depends on the order in which non-adjacent vertex pairs are chosen. Each Zykov-tree has exactly one branch that is exclusively generated by contractions. Later, Zykov's idea was described via graph homomorphism (see Chapter 5).

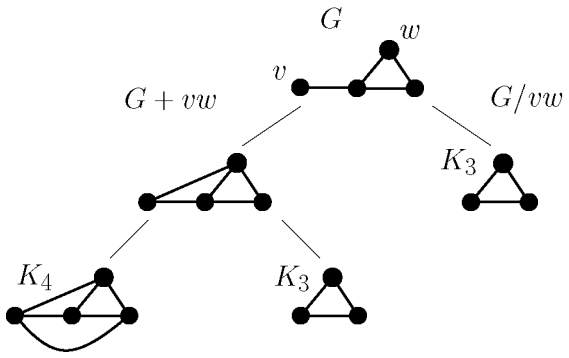


Figure 4.2: A Zykov-tree

Zykov's theorem itself does not give a colouring algorithm, but offers suggestions for its design. This is the basis of an algorithm by Corneil and Graham for $\chi(G)$, which searches through the Zykov-tree in a depth-first manner. Despite some technical refinements, this algorithm is inferior to the other sequential algorithms [37]. Today the use of the Zykov-tree in algorithm design has fallen into oblivion, but vertex contractions are applied in several other areas of graph theory and it has remained a powerful tool e.g. in proof by induction. Moreover, despite its rare usage today in the colouring process, this thesis was mainly motivated by Zykov's original idea.

4.4 Vertex contractions

Although, Zykov introduced his vertex contraction theorem [162] in the middle of the last century, it has not been applied much. There are a numerous applications of vertex contractions in the literature, but usually not for colouring ([14; 19; 29; 43; 45; 63; 113; 132; 146; 147]). The term of 'contraction' may have aliases such as 'merge', 'identification', 'gluing', 'fusing', 'amalgamating' or 'coalescing'. The latter is commonly used in the domain of register allocation problems ([19; 63; 113; 132; 146]). Coalescing is a terminology frequently used when two registers are coalesced where this is safe, in order to eliminate move operations between distinct variables (registers). Register allocation can be modelled as a graph colouring problem too. If the problem is represented by graph colouring, coalescing is a contraction of unconnected vertices. The purpose of merging may either be the simplification or the fusing of several simple graphs into one larger graph [112]. The vertex contraction technique is most helpful in proof by induction on the number of vertices or edges in a graph, where we can assume that a property holds for all contractions of a graph, and we can use it to demonstrate this for the larger graph. Usually, algorithms use vertex merging for **graph simplification** and for combination graphs. For instance, a simplification is done by merging two or more unconnected vertices to get fewer vertices before or during colouring. In [29], [147] and [43] a pre-processing of graphs is performed before colouring, where two vertices in a graph are merged to one if they are of the same colour in all colourings. This is analogous to studies of the development of a backbone or spine in the satisfiability problem [11; 126]. Here, the application of merging refers to removing one of two unconnected vertices. In fact, we also could remove edges that belong to the removed vertex. The only reason for performing these merges is to remove unnecessary or unimportant vertices from the graph in order to make it simpler. Those vertices that fulfil some specific condition will be removed from the data structure which describes the graph. This process will result in a loss of information. The second approach is to consider two graphs, that have certain colouring properties. For example one property might be that they are not k-colourable. Then the two graphs are joined by merging vertices from both graphs to create a more complex graph, where the desire is that the original properties are inherited. In both cases the identified vertices get the same colour. A nice example is the **Hajós construction** [76; 112] where $k-$uncolourable graphs are built from building blocks. One of the construction steps is a vertex contraction which may join building blocks.

Although, we shall be concerned with merging unconnected vertices, the **Hadwiger conjecture** [74] deserves a mention which is "one of the deepest unsolved problems in graph theory" [15]. The conjecture can be defined by edge-contractions where connected vertices are merged together by deleting the edge between them. The conjecture refers to graph colouring. Namely, each k-colourable graph contains K_k, a complete graph on k vertices as minor; that is, G has a sub-graph for which a K_k is reachable by applying edge-contractions. An equivalent form of the Hadwiger conjecture (the reverse form of that stated above) is that if there is no sequence of edge-contractions that brings graph G to the complete graph K_k, then G must have a vertex colouring

with $k-1$ colours. In spite of the different terms, from here on the terms 'contraction' and 'merge' will be used where applicable. 'Contraction' is a widely used expression in the literature, but so is 'merge'. The following models are based on the vertex contractions/merges of vertex-related structures such as the appropriate rows of the adjacency matrix. The name 'contraction' characterises well the identification of two objects as one, hence identification reduces the size of the graph. Nevertheless, the name 'contraction' does not appropriate term to describe operations on related structures of vertices so the term 'merge' seems more suitable because there can be no conventional shrinking in an associated graph.

4.5 Summary

This chapter discussed some important real-life applications of graph colouring and provided graph instances from different sources. We discussed several well-known graph colouring algorithms and described various approaches to solve the Graph Colouring Problem. Zykov's theorem introduces a new aspect, where colours are no longer needed to define and handle the problem. It implies a generalisation of the colouring and can be expressed via a graph homomorphism, where the vertices of a graph are mapped to vertices in another graph instead of mapping colours.

The contents of this thesis is supported by graph homomorphisms as well, therefore we keep separated chapter (Chapter 5) for them. In this thesis we generalise the Zykov's approach by introducing different models (Merge Models). We will demonstrate the novel models efficiency via a theoretical and experimental analysis as well. Merge Models reformulate the original problem, In this reformulated environment three different general frameworks will be introduced to describe an abstraction for algorithms based on the Merge Models. They provide a uniform and compact way in which algorithms can be defined. Embedding algorithms in the framework supports both their structural and performance comparison in a common basis, which can be anyway problematic. Traditional colouring schemes can be identified in one of the frameworks and extended schemes may be provided. The framework itself generalises the formal sequential colouring approach. Due to this generalisation such an embedding an algorithm can be enhanced, resulting in new algorithms. The novel aspect of the Merge Models implies the development of novel colouring strategies, i.e Merge Strategies. The Merge Models describes special graph homomorphisms, hence their analysis may reveal connections between strategies and different graph properties. Many novel efficient Merge Strategies will be provided which outperform several standard benchmark algorithms. Moreover, a general strategy design will be described which allows the application of machine learning techniques in the algorithm design.

Chapter 5

Graph homomorphism

The problem of k-colouring has another interpretation by using graph homomorphisms. In fact, we can generalise the k-colouring problem. The main benefit of the homomorphism approach is that we can get rid of the colours and we can design pure graph algorithms exploiting properties of the graphs stem from the description of particular homomorphisms. This section describes how we can make equivalence between k-colouring and certain graph homomorphisms.

5.1 H-colouring

Let H be a fixed graph. The homomorphism problem for H asks whether a graph G admits a homomorphism to H. A homomorphism of G to H is also called as *H-colouring* of G.

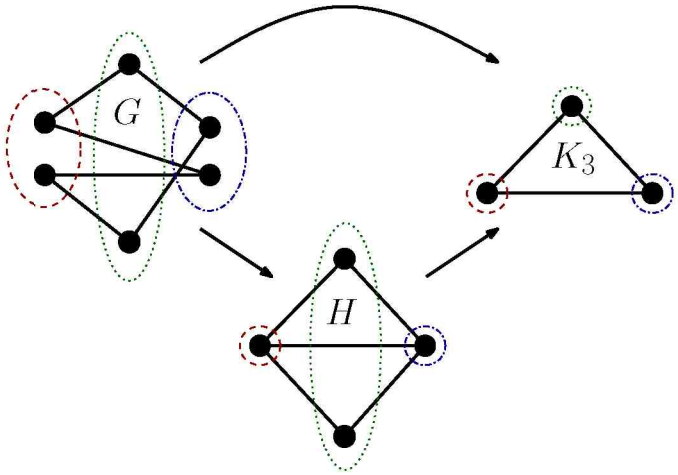
Definition 5.1 (H-colouring) *Let G and H be graphs. A homomorphism of G to H is a map $h : G \to H$, where we map vertices $V_G \to V(H)$ such that $\{x, y\} \in E_G \to \{h(x), h(y)\} \in E(H)$.*

If there exists a homomorphism $h : G \to H$ we shall write $G \to H$ and $G \nrightarrow H$ means there is no such homomorphism. If $G \to H$ we shall say G is homomorphic to H or that G is H-colourable. Note that the map is not necessarily surjective.

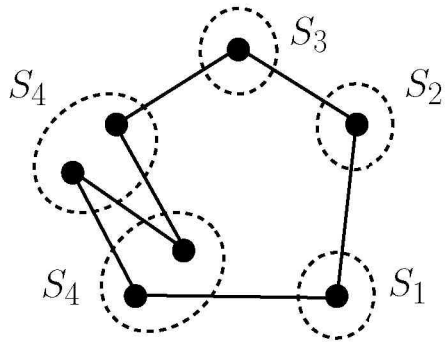
Definition 5.2 (Complete H-colouring) *Complete H-colouring, if exists, is a surjective H-colouring $G \xrightarrow{sur} H$.*

Composition $h_1 \circ h_2$ of homomorphism $h_1 : G \to H_1$ and $h_2 : H_1 \to H_2$ is homomorphism of $G \to H_2$ (see Figure 5.1(a)). Compositions will play an important role in the design of sequential colouring algorithms using homomorphisms, in this case consecutive homomorphisms will substitute each colouring steps.
Although, the substitution will bring several benefits, we cannot avoid the complexity of the k-colouring problem. The following theorem shows how hard it is to find a homomorphism between two graphs.

Theorem 5.1 (Hell and Nešetřil, 1990 [85]) *If H is bipartite or contains a loop, then H-colouring is polynomial time solvable; otherwise, H is $\mathcal{NP}$-complete.*



(a) The procedure for composing homomorphisms



(b) Vertex set partition of a H-colouring

5.2 H-colouring and k-colouring

From the above definition it is clear that homomorphism preserves the adjacency relation and G admits an H-colouring if and only if there is a partition of V_G into sets S_i so that each of them is an independent set and there are no edges between the vertices of S_i and S_j if $\{S_i, S_j\} \notin E(H)$. The S_i sets represent the vertices of H (as illustrated in Figure 5.1(b)). Recall that a k-colouring of G is a mapping $c : V_G \to \{1, 2, \ldots, k\}$, where adjacent vertices have distinct colours, which means that $c(u) \neq c(v)$ whenever $\{u, v\} \in E_G$. Hence, colour classes $\{S_i\}_{i=1}^{k}$ form independent sets and there is an edge between any two colour classes S_i and S_j if and only if their components are connected. If $k = \chi$, then there are no colour classes so S_i and S_j are unconnected, otherwise we could decrease the number of colours used, applying a common colour for their members. The S_i-s form a vertex set of a complete graph. However, for larger k than χ we can get unconnected S_i-s, e.g. for $k = n$, but here we should notice that the condition $c(u) \neq c(v)$ is equivalent to the condition $\{c(u), c(v)\} \in E(K_k)$; that is, we can embed graphs defined by colour classes into a complete graph and we may conclude the following.

Proposition 5.1 *Homomorphisms $h : G \to K_k$ are precisely the k-colourings of G.*

As mentioned above, the embedding of a graph defined by colour classes does not necessarily result in a complete graph, but for those special cases when the result is a complete graph the homomorphism will be a complete H-colouring.

Definition 5.3 (Complete k-colouring) *Complete H-colouring, if it exists, is a complete K_k-colouring.*

Each complete k-colouring of G is associated with a partition of vertices into k non-empty independent sets, any two of which are joined by at least one edge.

5.3 Chromatic and Achromatic number

Theorem 5.2 (Colour Interpolation Theorem [86]) *If a graph admits a complete k_1-colouring and k_2-colouring then it admits a complete colouring for all k, where $k_1 \leq k \leq k_2$.*

The smallest k where the graph G admits a k-colouring defines the chromatic number of G (Eq. 5.1) and the largest k where the graph G admits a *complete* k-colouring defines the achromatic number of G (Eq. 5.1). Note that any χ-colouring of a graph must be complete.

$$\chi(G) = \min_k\{k \mid G \to K_k\} \tag{5.1}$$

$$\psi(G) = \max_k\{k \mid G \xrightarrow{sur} K_k\} \tag{5.2}$$

Thus we may conclude from the Colouring Interpolation Theorem that G admits a complete k-colouring for any k between its chromatic and achromatic number.

Proposition 5.2 *Let G be a graph. For each k, $\chi(G) \leq k \leq \psi(G)$, G admits a complete k-colouring.*

It is not hard to verify that if $G \to H$ then $\chi(G) \leq \chi(H)$. Indeed, if $H \to K_k$ exists then G is K_k-homomorphic thanks to the composition of homomorphisms. Consequently if $\chi(G) > \chi(H)$, then $G \not\to H$. We can similarly prove that if $G \to H$, then $\omega(G) \leq \omega(H)$, using a $K_k \to G$ homomorphism, based to the following equation:

$$\omega(G) = \max_k\{k \mid K_k \to G\}$$

5.4 Summary

In this chapter we saw how the $H-$colouring problem generalises the traditional k-colouring problem. The k-colouring was interpreted as a homomorphism. The creation of such a map is not easy for any kind of target graph. Nevertheless, the complexity of the H-colouring remains the same for k-colouring, but the benefits of creating of graph homomorphisms instead of colour assignments can be exploited. On the one hand, as only one structure is necessary for the graph, we can omit the colours. On the other hand, using the possibility of homomorphism compositions, we can transform a graph into other graph instances which can tell us more about the structure of the original problem. Moreover, we can generate a homomorphic graph series between a graph and a complete graph by successive homomorphisms (see Figure 5.1(a)). These graph series or consecutive homomorphisms correspond to particular sequential colourings. In the next chapter we will present different approaches for achieving homomorphisms like this for $k-$colouring. Although, [16; 107; 148; 159]. describes how we can define other graph colourings such as circular and fractional colourings through H-colouring, defining various target graphs. Furthermore, H-colouring can be analogously stated for any relational system H, e.g. for the general constraint satisfaction problem.

Chapter 6

Quotient and Power methods

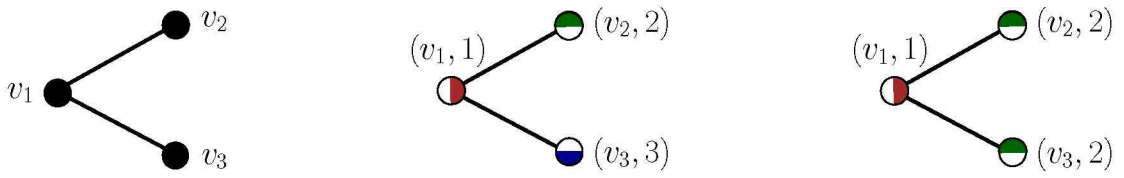
In this chapter we shall define graph colouring processes as a series of homomorphisms using quotient or power graphs, where the vertices which get the same colour will be 'glued' or 'grouped' together, respectively, to form a new vertex set. Here a modified vertex set usually results in a modified edge set as well.

These graph operations produce helpful graph structures which can be exploited for an efficient colouring and also help provide a deeper insight into the colouring procedure. Moreover, they allow us to design efficient new or redesign existing graph colouring algorithms in a framework supported by quotient or power graphs (see Juhos et al. [96–102]).

In the following we shall introduce the theory of quotient and power methods and later on we shall discuss the implementation details by describing current and novel colouring methods.

6.1 Motivation

Figure 6.1(c) shows a drawing of a graph where the vertices are denoted by circles, while Figures 6.1(d) and 6.1(e) show different proper colourings of the same graph, namely a 3-colouring and a 2−colouring. Although, colours and vertices are different entities, they may be jointly encoded in one object by a circle symbol. Colour entities are implicitly encoded in the vertex, but they can be handled separately. In Figure

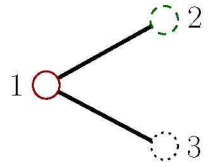


(c) A graph (d) 3−colouring (e) 2−colouring

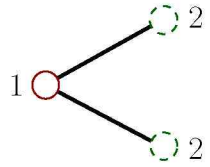
Figure 6.1: Different colourings of a graph. Vertices $\{v_1, v_2, v_3\}$ and colours $\{1, 2, 3\}$ are not separate entities. One circle encodes information about a graph vertex and a colour as well.

6.2 colours have been detached from the vertices. Figures 6.2(a) and 6.2(b) display

the detached colours and their relations in accordance with the $3-$colouring and the $2-$colouring of Figure 6.1(d) and 6.1(e), respectively. The detached $2-$colouring clearly shows the redundancy, of the colour 2 instances in Figure 6.2(c). In order to eliminate this redundancy one can eliminate the different instances of the colour 2 and use just one instance instead, this elimination leads to a compact representation of the colour relations. If the elimination step is stored, then this compact representation can define the original $2-$colouring. Since the graph of the detached colours inherits the original vertex relations, this graph is equivalent to the original one. Consequently, colours and vertices can be identified as a common entity. Depending on the context where they occur, this entity can be called either as a colour or vertex. Usually, it is reasonable to call them as a vertex because colours can assist the presentation and the general explanation for instance colours can be useful progress indicators of a colouring process, where coloured and uncoloured vertices are distinguished.



(a) Relation between the colours in a $3-$colouring.

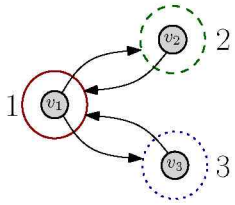


(b) Relation between the colours in a $2-$colouring. Colour 2 has two instances. *Redundancy.*

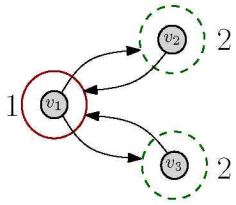
(c) Relation between the colours in a $2-$colouring. Colour 2 has one instance. *No redundancy.*

Figure 6.2: Motivation of quotient graphs. Colours $1, 2$ and 3 are circles. $2-$colouring introduce choices: keeping redundant colour instances or eliminating redundancy.

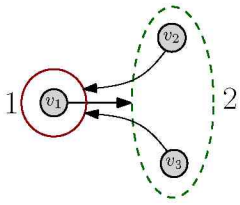
The compact representation of the colouring requires storing of each elimination step to have a chance of recovering the colouring of the original graph. To eliminate this storing process, first preserve the difference between vetrices and colours, and handle the vertex – colour relations together with the eliminations. Colours and vertices should be detached as well, but both are retained as different entities, as illustrated in Figure 6.3. Vertices are the inner circles while colours are the outer circles. This special positioning of the circles is just to aid understanding, but they could be arranged in other ways. The vertex – colour assignments are represented by directed edges from inner circles to the outer ones. Here, the 2-colouring provides the possibility of the elimination as well. The two instances of the colour 2 may be eliminated by allowing only one single instance of the colour 2. Figure 6.3(b) shows a colour redundant representation of the $2-$colouring of Figure 6.1(e). But in Figure 6.3(c) the colour redundancy has been eliminated. This approach provides a compact representation for the colours and hence the vertex–colour relations. Nevertheless, there is no need to store the elimination steps, since the vertex – colour relations, i.e. the vertex – colour assignments are always available. The created graph (Figure 6.3) can be transformed into the graph of Figure 6.2(c) by contracting the outer circle, as depicted in Figure 6.4.



(a) Vertex – colour relations in a $3-$colouring.

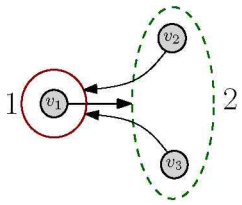


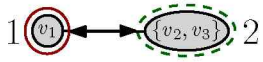
(b) Vertex – colour relations in a $2-$colouring. Colour 2 has two instances. *Redundancy.*



(c) Vertex – colour relations in a $2-$colouring. Colour 2 has one instance. *No redundancy.*

Figure 6.3: Motivation behind power graphs. Vertices v_1,v_2 and v_3 are inner circles, while colours $1, 2$ and 3 are outer circles. The figures depict vertices – colour relations. $2-$colouring introduce choices: keeping redundant colour instances or eliminating redundancy.





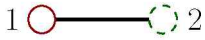


Figure 6.4: Contraction of outer circles.

The approach which keeps the vertex – colour relations is the motivation behind the power methods, while the vertex – colour identification is the basis of the quotient methods.

6.2 Quotient method

Definition 6.1 (Topological space [155]) *A topological space is a set V together with $\mathcal{V}$, a set of subsets of V, satisfying the following axioms: the empty set and V are in $\mathcal{V}$; the union of elements $\mathcal{V}$ is also in $\mathcal{V}$; any finite intersection of elements of $\mathcal{V}$ is also in $\mathcal{V}$.*

The set $\mathcal{V}$ is called a topology on V. A quotient space comes from the original one by 'gluing' the elements of the space. More precisely

Definition 6.2 (Quotient space [155]) *Let V be a topological space and $\sim$ be an equivalence relation on V. The topological quotient space $V/\sim$ is composed of equivalent classes of the space V by relation $\sim$, using a surjective map $V \to V/\sim$.*

Equivalence classes form a partition, conversely, a partition defines an equivalence relation $\sim$ which is the kernel of the surjective map [1]. If only one equivalence class has two or more elements, then that class describes the whole partition, i.e. the relation

[1] The kernel of a function f is $\ker f = \{(v, u) | f(v) = f(u)\}$

$\sim$. Let S be a subset of V, where $v \sim u$ iff $v, u \in S$. Then following [78] and [125], we may denote $V/\sim$ by V/S as well or by using elements of S, e.g. in the case of $S = \{v, u\}$ we can also use notation V/vu.

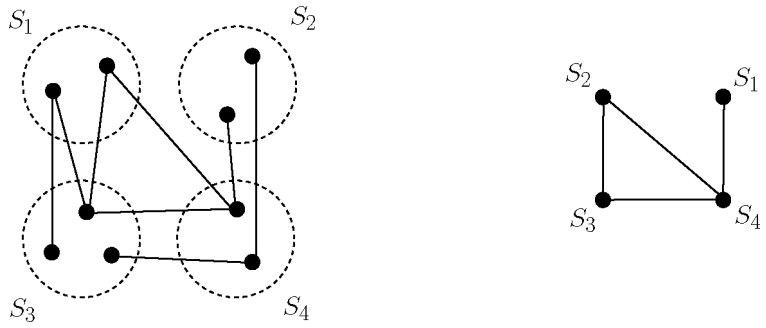
According to [119], the form of the previous definition for a particular case, namely for graphs, is the following

Definition 6.3 (Quotient graph) *Given a graph $G = (V, E)$ and a partition[2] $\mathcal{S}$ of V, the quotient graph $G/\mathcal{S}$ is the graph $(\mathcal{S}, \mathcal{E})$ where $\mathcal{E} = \{\{S_i, S_j\} \mid S_i \times S_j \cap E \neq \emptyset\}$.*

In [94; 100–102] the author described a general model where the graph colouring is efficiently modelled by special Quotient graphs, forming a general Quotient method for the graph colouring. They showed that efficient graph colouring algorithms could be designed based on the Quotient method.

The following relations can be identified between images of H-colourings and Quotient Graphs, as described in Section 5.1.

Proposition 6.1 (H-colourings and Quotient Graphs) *Every quotient graph of G is a homomorphic image of G and, conversely, every homomorphic image of G is isomorphic to a quotient of G.*



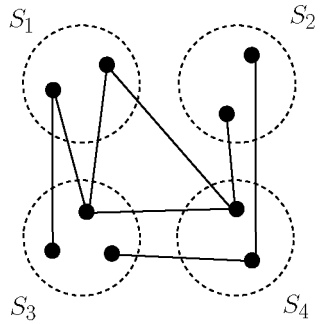
(a) A graph G with partition $\mathcal{S}$

(b) $G/\mathcal{S}$ quotient graph

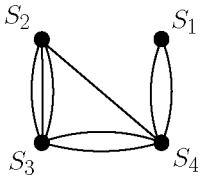
Figure 6.5: An example of a quotient graph.

A quotient graph (see Figure 6.5) is a simple graph, thus its edges form a set, but retaining different images of the original edges can lead to multiple edges between classes and induce an edge multiset in a quotient multigraph (see Figure 6.6) in accordance with Def. 2.12. To distinguish between quotient graph and multigraphs, we shall use a double slash in our notation for quotient multigraphs e.g. $G /\!/ \mathcal{S}$. Quotient graphs may be constructed by graph vertex contractions, where each S_i is a set of contracting vertices. Recall that vertex contraction can be applied to connected and unconnected vertices as well, but we will use edge-contraction for contracting two connected vertices. As mentioned earlier, unless otherwise stated vertex contraction will be used for unconnected vertices only.

[2] $\mathcal{S} = \bigcup S_i$ and $S_i \cap S_j = \emptyset$ if $i \neq j$



(a) A graph G with partition $\mathcal{S}$



(b) $G /\!/ \mathcal{S}$ quotient multigraph

Figure 6.6: An example of producing quotient multigraph.

Graph vertex colouring defines partition of vertices, where the same colour vertices form the equivalent classes and, conversely, any partition provides a colouring. That is, $x, y \in S_i$ implies $\{x, y\} \notin E$ to get a valid graph colouring. To simplify the structure of the graph we can create a quotient graph by merging vertices in the same class. Applying a vertex contraction for each colouring steps results in several intermediate quotient graphs until a complete graph is obtained. We shall see how beneficial the application of the vertex contration is in the graph colouring. In minimum colouring, we have to find a homomorphism which results in as small complete graph as possible. Thus an algorithm must look for the longest sequence of contractions; that is, the longest homomorphic graph series, because each merge decreases the number of vertices in the graph, hence the longest path results in the smallest graph.

Though contracted graphs specify a H-colouring process quite well, they lose information about the original graph structure when we simplify it via contractions. We will introduce another method which keeps information about the original graph and works in harmony with the H-colouring principle as the quotient method does.

6.3 Power method

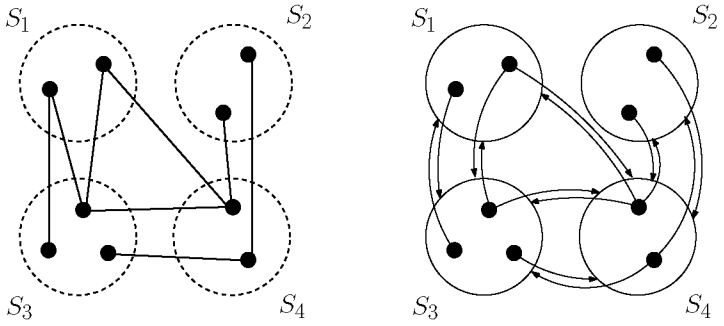
Instead of contracting or merging vertices we can merge related structures of vertices to produce a special non-quotient graph. E.g. merging relevant rows of the adjacency matrix gives rise to a vertex 'grouping' effect. This grouping can be characterised by power graphs (see Figure 6.7), which put putting a new vertex called 'group-vertex' (which encompasses some of the original vertices) into the original vertex set. A group-vertex takes over the incoming edges from the encompassed vertices. A group-vertex will be a colour class in the traditional sense, hence all vertices belonging to a group-vertex may be regarded as coloured vertices with the same colour. A power graph can be defined on a power set of the vertices of a graph in accordance with [2].

Definition 6.4 (Power graph) *Let $G = (V, E)$ be a graph. The vertices of a power graph $G' = (V', E')$ are defined by a subset of the power set of the G vertices $V' \subseteq 2^V$.*

Power vertices are connected to each other by power edges $E' \subseteq V' \times V'$.

The author introduced the Power method for the graph colouring problem in [96], where the graph colouring is modelled by a special Power graph sequence. They demonstrated the efficiency of the Power method and developed several powerful graph colouring algorithms based on the method described in [97–100].

Figure 6.7 shows how a power graph may be created from a partition of vertices. The new vertex set is a subset of the power set of the original vertices, where we can find group-vertices that represent equivalent or colour classes. The original graph is a simple graph which defines its edge set as a symmetric relation; if $(x, y) \in E$ then $(y, x) \in E$. We can make these undirected edges as combinations of two directed edges, where one is from a vertex to one of its neighbours and another is the reverse case. The group-vertices become new endpoints of the directed edges that determines a vertex-'neighbour colour' relation. Therefore we can map two directed edges representing an undirected edge in the original graph to two power edges of the power graph. This map is surjective, but not necessarily injective. For example, if neighbours of a vertex have the same colour, then four directed edges are mapped to three power edges, like vertices in S_1 and S_3 in Figure 6.7 In order to get an injective edge map we have to



(a) A graph G with partition $\mathcal{S}$

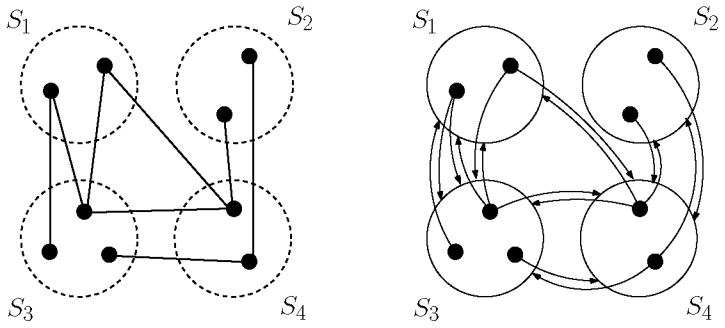
(b) $G^{/\mathcal{S}}$ power graph

Figure 6.7: An example of producing power graph.

use a power multigraph with a multiset for its edge set, as we saw earlier in the case of quotient multigraphs in accordance with Def. 2.12. A similar graph homomorphism can be defined between a graph which is equivalent with the original graph [3] and its colouring power graphs. Hence, we shall denote it in harmony with the quotient graphs, but use superscript to represent the power. Thus, denote H-colouring power graphs by $G^{/\mathcal{S}}$ and, likewise, multigraphs by $G^{/\!/\mathcal{S}}$.

Note that contracting the appropriate power vertices (with all the vertices it encompasses, e.g. S_1 and its two vertices in Figure 6.7), results in a homomorphism from a

[3]Vertices in the related graph are doubled or interpreted as a (vertex,colour–vertex) pair. Initially each vertex gets different colours, then some of them can share the same colour–vertex; that is, certain colour–vertices get contracted. An outgoing edge connects a vertex with a colour-vertex.



(a) A graph G with partition $\mathcal{S}$

(b) $G^{/\!/\mathcal{S}}$ power multigraph

Figure 6.8: An example of producing power multigraph.

power graph to a quotient graph .

With our sequential graph colouring approaches we can construct a colouring for a graph by progressively merging structures of a graph that generates quotient or power graph sequences. The final graph in the sequence defines a colouring where the contracted or grouped vertices get the same colour in the original graph. In fact, quotient or power graph sequences themselves define the whole colouring process. Here the use of colours for vertices of intermediate graphs was employed in order to see the steps involved.

6.4 Summary

This chapter introduced two methods, namely the Quotient and Power methods for the graph colouring problem and, in addition, two variants of them. These methods model the graph colouring problem via certain graph homomorphisms. The composition of several homomorphisms defines colouring steps in the traditional sense.

In the next chapter we shall provide a matrix representation of these models with special matrix operations, which results in homomorphic images based on the Quotient and Power methods.

Chapter 7

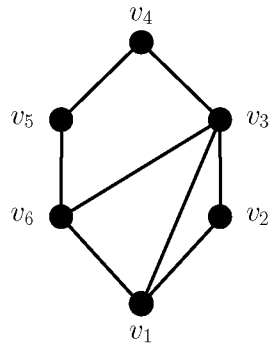
Merge Models

The relation between the original graph and a quotient or power graph/multigraph is defined by a graph homomorphism. The author introduced four kinds of matrix operations, called Merge Operations to map a representation structure of the original graph to its four different homomorph images, respectively, and then subsequent Merge Operations will produce vertex colouring [96; 100]. They showed that Merge Operations produce appropriate homomorph images of the original problem in accordance with Chapter 5, modelling the original graph colouring problem [96; 100]. The representations and the operations form new colouring models, called Merge Models, that supports parallel implementations. They got significant improvements both theoretically and via experiments in [99] when an algorithm applied their models. Exploiting the performance they designed powerful graph colouring algorithms in [94; 97–99; 101; 102]. The details of their analysis can be found in chapters 10 and 11.

Vertices having the same colour in the traditional colouring process induce merges in the adjacency matrix, and each edge that is connected to these vertices is either collapsed into a single edge or forms a multi-edge in the resulting structure. Multi-edges can be identified as single but weighted edges where their weight counts the multiplicity of the edge. We shall present matrix representations of the result quotient or power graphs. There will be two subtypes of representations where one does not depend on the number of collapsed edges, while the other one does. These representations are used together with basic Merge Operations to create power graphs where only rows are merged or produce quotient graphs when the relevant columns are also merged. By combining these representations with the Merge Operations we will provide four colouring models called the Binary/Integer Merge Square and the Binary/Integer Merge Table models. Their representation matrices will be denoted by $A, \mathbb{A}, T$ and $\mathbb{T}$, respectively. Here, Merge Squares are associated with the adjacency matrix of the merged graph, i.e. a quotient graph. The Integer types assign weights to the edges according to how many edges are merged. The Binary types approach simply collapses these edges onto the same one common edge, but it does not preserve their cardinality. The tables track information about the original vertices, while the squares omit. The graph depicted in Figure 7.1 and its adjacency matrix will be used as examples to help explain the different model types.

7.1 Merge matrices

In order to generate sequential colourings, consecutive homomorphisms will be applied starting with the original graph and ending up with a complete graph of a graph which is homomorphic with a complete graph. The number of transformation steps can be followed by the upper index t e.g. $\mathbb{A}^t$.



	v_1	v_2	v_3	v_4	v_5	v_6
r_1	·	1	1	·	·	1
r_2	1	·	1	·	·	·
r_3	1	1	·	1	·	1
r_4	·	·	1	·	1	·
r_5	·	·	·	1	·	1
r_6	1	·	1	·	1	·

Figure 7.1: A graph G and its initial Merge Matrix, the adjacency matrix: the v-s refer to vertices and the r-s refer to rows, i.e. colours. The 0-s have been replaced by dots for the sake of clarity.

The initial Merge Matrix is the adjacency matrix of G: $A^{[0]} = \mathbb{A}^{[0]} = T^{[0]} = \mathbb{T}^{[0]} := A$. Here we shall only deal with valid colourings, hence simple non-adjacent vertices can be merged together. In the case of Merge Squares representations, a Merge Square is the unweighted or weighted adjacency matrix of a quotient graph, thus columns and rows refer to the same objects of the graph, namely to the merged vertices/colour classes. The condition of the merge depends on the relation between vertices, i.e. the edges of the quotient graph. The coincidence of a given row and column of the Merge Square must be zero. We can easily see that this condition is the same for Merge Tables (MT), but it breaks the symmetry of the representation. Therefore, we have to check the adjacency between a normal the original vertex (which refers to an MT column) and a merged vertex-set/colour classes (which refers to an MT row). We can summarise the **merge conditions** by the following:

$$a_{ij}^{[t]} = \mathrm{a}_{ij}^{[t]} = t_{ij}^{[t]} = \mathbb{t}_{ij}^{[t]} = 0 \tag{7.1}$$

Consequently $a_{ji}^{[t]} = \mathrm{a}_{ji}^{[t]} = 0$ and thanks to the inherited graph property $a_{ii}^{[t]} = a_{jj}^{[t]} = \mathrm{a}_{ii}^{[t]} = \mathrm{a}_{jj}^{[t]} = 0$. Next we shall define the following matrices:

$$P = I_i \otimes I_j \qquad R = I_j \otimes I_j \qquad W = P - R \tag{7.2}$$

where I_i is the $i-$th row of the identity matrix, P (Plus) will be used for addition (or bitwise–OR operation) of the j-th row of a matrix to the corresponding i-th row. R (Reduction or Minus) will support the subtraction of the j-th row from itself, thereby setting its components to zero. This could also be done by a bitwise exclusive or (XOR). In the case of the third matrix, W combines these operations together. Here let a and b define the i-th and j-th **row vector** of a matrix for step t. We now define the four models formulated both as row/column operations and matrix manipulations. First the

integer-based models and then the binary-based model, which do not track the number of edges folded into an edge.

7.1.1 Merge Tables

With addition or bitwise-OR two rows of an adjacency matrix creates a power graph structure, which characterises a relation between the original vertices and the neighbouring colours or colour classes. We may associate rows of an adjacency matrix to colour classes or power vertices and columns to vertices of the original graph. The matrices of these power graphs are known as Merge Tables owing to their shape. As previously mentioned, there are two subtypes, namely a weighted type and an unweighted type, based on how the number of the edges are taken into account in the merging process.

The Integer Merge Table

Integer Merge Tables keep track of multi-edges. As we said earlier we can refer to a multi-edge by a weight, counting the number of edges folded into one edge during the merging process.

A row-based formulation of the i-th and j-th row of $\mathbb{T}$ after merging the j-th vertex into the i-th: let $\mathbb{t}_i$ be the i-th row and $\mathbb{t}\_{}_i$ be the column vector. Then

$$\mathbb{T}_i^{[t+1]} = \mathbf{a} + \mathbf{b} \quad , \quad \mathbb{T}_j^{[t+1]} = \mathbf{b} - \mathbf{b} = 0 \tag{7.3}$$

A matrix-based formulation

$$\mathbb{T}^{[t+1]} \quad = \mathbb{T}^{[t]} + W\mathbb{T}^{[t]} = (I + W)\mathbb{T}^{[t]} \tag{7.4}$$

where W is defined in Eq. 7.2. In Figure 7.2, rows r_1 and r_4 have merged, after which the row r_4 is removed to get a collapsed Merge Table. An Integer Merge Table model does not lose any edge from the original graph because it keeps track them as multiple edges. Multiple edges are represented by values which are greater than one. It occurs when two rows have non-zero elements in the same positions in the merge. In Figure there is a 2 in the $(\{r_1, r_4\}, r_3)$ position of the collapsed Merge Table. This value of 2 appears as a multiple edge which starts from v_3 to the only power node $\{v_1, v_4\}$ in Figure 7.2(c). Due to this fact, the sum of the matrix does not change. We could normalise entries of Integer Merge Tables in several ways, one can be the leaving out the counting the number of edges folded together to get a $\{0, 1\}$ binary matrix.

The Binary Merge Table

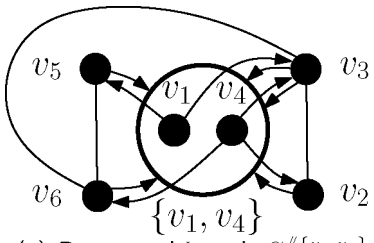
The binary version of the Merge Tables focuses on the relation but not the degree of the relation between edges and colours. Here we have two options; apply a piecewise OR operation (see equations 7.5 and 7.7) or apply Integer Merge Table model and subtract

	v_1	v_2	v_3	v_4	v_5	v_6
r_1	·	1	1	·	·	1
r_2	1	·	1	·	·	·
r_3	1	1	·	1	·	1
r_4	·	·	1	·	1	·
r_5	·	·	·	1	·	1
r_6	1	·	1	·	1	·

(a) Adjacency matrix A_G

	v_1	v_2	v_3	v_4	v_5	v_6
$\{r_1, r_4\}$	·	1	2	·	1	1
r_2	1	·	1	·	·	·
r_3	1	1	·	1	·	1
r_4	·	·	·	·	·	·
r_5	·	·	·	1	·	1
r_6	1	·	1	·	1	·

(b) Integer Merge Table $\mathbb{T}(G^{/\!/\{v_1,v_4\}})$



(c) Power multigraph $G^{/\!/\{v_1,v_4\}}$

	v_1	v_2	v_3	v_4	v_5	v_6
$\{r_1, r_4\}$	·	1	2	·	1	1
r_2	1	·	1	·	·	·
r_3	1	1	·	1	·	1
r_5	·	·	·	1	·	1
r_6	1	·	1	·	1	·

(d) Collapsed Integer Merge Table $\mathbb{T}(G^{/\!/\{v_1,v_4\}})$

Figure 7.2: Merging (addition) rows r_1, r_4 of A_G results in a $\mathbb{T}(G^{/\!/\{v_1,v_4\}})$ Merge Table.

irrelevant items from it. The latter solution may be useful in algebraic methods (see equations 7.6 and 7.8), while the former is easy to implement.

A row-based formulation

$$T_i^{[t+1]} = \mathbf{a} \vee \mathbf{b} \quad , \quad T_j^{[t+1]} = 0^T \tag{7.5}$$

$$T_i^{[t+1]} = \mathbb{T}_i^{[t+1]} - \mathbf{a} \circ \mathbf{b} \quad , \quad T_j^{[t+1]} = 0^T \tag{7.6}$$

$$\mathbf{a} \circ \mathbf{b} = diag(\mathbf{a} \otimes \mathbf{b}) = \sum_i (\mathbf{a} \otimes \mathbf{b}) I_i$$

A matrix-based formulation

$$T^{[t+1]} = T^{[t]} \vee PT^{[t]} - RT^{[t]} \tag{7.7}$$

$$T^{[t+1]} = \mathbb{T}^{[t+1]} - \sum_j (\mathbf{a} \otimes \mathbf{b})(I_j \otimes I_i) \tag{7.8}$$

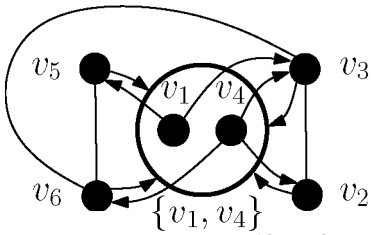
where P and R are defined in Eq. 7.2. In Figure 7.3, row r_4 is merged with row r_1 to form $\{r_1, r_4\}$, after which r_4 is deleted. An option to get a Binary Merge Table from the integer counterpart can be that when each non-zero elements are multiplied by the reciprocal value of the element. Unfortunately, this piecewise operation does not support well the algebraic computation, nevertheless it can be useful in the practical implementation. A Merge Table describes relation between vertices and colour classes, but a power graph can be transformed into an appropriate quotient graph of the original graph by contractions as seen in Figure 6.3. Then vertices and power vertices/colour classes can be identified only one object either vertex or colours depending on the context where we would like to use them. The application of a Merge Operation to

	v_1	v_2	v_3	v_4	v_5	v_6
r_1	·	1	1	·	·	1
r_2	1	·	1	·	·	·
r_3	1	1	·	1	·	1
r_4	·	·	1	·	1	·
r_5	·	·	·	1	·	1
r_6	1	·	1	·	1	·

(a) Adjacency matrix A_G

	v_1	v_2	v_3	v_4	v_5	v_6
$\{r_1, r_4\}$	·	1	1	·	1	1
r_2	1	·	1	·	·	·
r_3	1	1	·	1	·	1
r_4	·	·	·	·	·	·
r_5	·	·	·	1	·	1
r_6	1	·	1	·	1	·

(b) Binary Merge Table $T(G^{/\{v_1,v_4\}})$



(c) Power graph $G^{/\{v_1,v_4\}}$

	v_1	v_2	v_3	v_4	v_5	v_6
$\{r_1, r_4\}$	·	1	1	·	1	1
r_2	1	·	1	·	·	·
r_3	1	1	·	1	·	1
r_5	·	·	·	1	·	1
r_6	1	·	1	·	1	·

(d) Collapsed Binary Merge Table $T(G^{/\{v_1,v_4\}})$

Figure 7.3: Merging (bitwise OR) rows r_1, r_4 of A_G results in $T(G^{/\{v_1,v_4\}})$ Merge Table.

the rows and the relevant columns as well we arrive to quotient graph where vertices becomes colours and conversely.

7.1.2 Merge Squares

The result matrix after a merge of rows and appropriate columns is square, more exactly either weighted or unweighted adjacency matrix of the vertex contracted graph. Similar to the Merge Tables we will define their counterpart Merge Squares.

The Integer Merge Square

A row/column-based formulation let $\mathbb{a}_i$ be the i-th row and $\mathbb{a}_{\_i}$ be the column vector and define $\mathbb{a}_j$ and $\mathbb{a}_{\_j}$ in the same way for the j-th row and column.

$$\mathbb{A}_i^{[t+1]} = \mathbf{a} + \mathbf{b} \quad , \quad \mathbb{A}_j^{[t+1]} = 0^T \tag{7.9}$$

$$\mathbb{A}_{\_i}^{[t+1]} = \mathbf{a}^T + \mathbf{b}^T \quad , \quad \mathbb{A}_{\_j}^{[t+1]} = 0 \tag{7.10}$$

A matrix-based formulation

$$\mathbb{A}^{[t+1]} = \mathbb{A}^{[t]} + W\mathbb{A}^{[t]} + \mathbb{A}^{[t]}W^T \tag{7.11}$$

Since $\mathbb{a}_{ij}^t = 0$ and $\mathbb{a}_{ji}^t = 0$, it follows that $W\mathbb{A}^{[t]}W^T$=0. Due to this, we can rewrite

Eq. 7.11 as

$$\mathbb{A}^{[t+1]} = (I + W)\mathbb{A}^{[t]}(I + W)^T \tag{7.12}$$

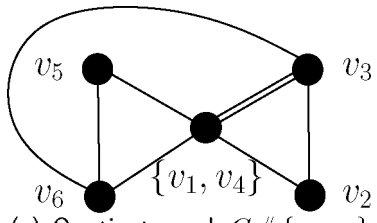
where W is defined in Eq. 7.2. Note that the condition of the merging of the row i and j is $\mathbb{t}_{\mathbb{i}\mathbb{j}} = \mathbb{t}_{\mathbb{j}\mathbb{i}} = \mathbb{t}_{\mathbb{i}\mathbb{i}} = \mathbb{t}_{\mathbb{j}\mathbb{j}} = 0$. Redefining the starting matrix according to $\mathbb{T}_0 = A - I$ allows us to keep track of which vertices have been encompassed by a merged vertex. The original Merge Operation should not be modified. In this case, -1 entries then refer to the merged vertices. We should also take this modification into account in the algorithm design. In Figure 7.4, a Merge Square has caused both columns and rows to be merged. The result is an adjacency matrix of the merged graph with weights on the edges that describe the number of edges that were merged.

	v_1	v_2	v_3	v_4	v_5	v_6
r_1	·	1	1	·	·	1
r_2	1	·	1	·	·	·
r_3	1	1	·	1	·	1
r_4	·	·	1	·	1	·
r_5	·	·	·	1	·	1
r_6	1	·	1	·	1	·

(a) Adjacency matrix A_G

	$\{v_1, v_4\}$	v_2	v_3	v_4	v_5	v_6
$\{r_1, r_4\}$	·	1	2	·	1	1
r_2	1	·	1	·	·	·
r_3	2	1	·	·	·	1
r_4	·	·	·	·	·	·
r_5	1	·	·	·	·	1
r_6	1	·	1	·	1	·

(b) Integer Merge Square $\mathbb{A}(G^{/\!/\{v_1,v_4\}})$



(c) Quotient graph $G /\!/ \{v_1, v_4\}$

	$\{v_1, v_4\}$	v_2	v_3	v_5	v_6
$\{r_1, r_4\}$	·	1	2	1	1
r_2	1	·	1	·	·
r_3	2	1	·	·	1
r_5	1	·	·	·	1
r_6	1	·	1	1	·

(d) Collapsed Integer Merge Square $\mathbb{A}(G^{/\!/\{v_1,v_4\}})$

Figure 7.4: Merging (addition) rows r_1, r_4 of A_G results in Integer Merge Square.

Note that the merge condition, in the case of the row i and j, is $\mathbb{a}_{\mathbb{i}\mathbb{j}} = \mathbb{a}_{\mathbb{j}\mathbb{i}} = \mathbb{a}_{\mathbb{i}\mathbb{i}} = \mathbb{a}_{\mathbb{j}\mathbb{j}} = 0$ in accordance with Eq. 7.1. Hence after a merge $\mathbb{a}_{\mathbb{i}\mathbb{j}} = \mathbb{a}_{\mathbb{i}\mathbb{j}} + \mathbb{a}_{\mathbb{j}\mathbb{i}} + \mathbb{a}_{\mathbb{i}\mathbb{i}} + \mathbb{a}_{\mathbb{j}\mathbb{j}}$ remains zero. We could also use up $\mathbb{a}_{\mathbb{i}\mathbb{i}}$ cells to store additional structural information. Starting with the $\mathbb{A}_0 = A - I$ matrix instead of the pure adjacency matrix A, we could then count the number of vertices encompassed by a merged vertex, while keeping the original Merge Operation [1] . In this case, diagonal entries will contain all the cardinalities. However, an algorithm should handle the modified diagonal elements. Similar to the Integer Merge Tables, the Integer Merge Square model does not lose any edges either, but store them as multiple edges. Thus, the sum of the matrix does not change in this model.

[1] $A + I$ is an alternative here.

The Binary Merge Square

A row/column-based formulation Let a_j be the j-th row and let a_j be the corresponding column vector. Then

$$A_i^{[t+1]} = \mathbf{a} \vee \mathbf{b} \quad , \quad A_j^{[t+1]} = 0^T \tag{7.13}$$

$$A_i^{[t+1]} = \mathbb{A}_i^{[t+1]} - \mathbf{a} \circ \mathbf{b} \quad , \quad A_j^{[t+1]} = 0^T \tag{7.14}$$

$$A_{\_i}^{[t+1]} = (A_i^{[t+1]})^T \quad , \quad A_{\_j}^{[t+1]} = 0 \tag{7.15}$$

A matrix-based formulation

$$A^{[t+1]} = A^{[t]} \vee (PA^{[t]} + A^{[t]}P^T) - (RA^{[t]} + A^{[t]}R^T) \tag{7.16}$$

$$A^{[t+1]} = A^{[t]} \vee (PA^{[t]}P^T) - (RA^{[t]}R^T) \tag{7.17}$$

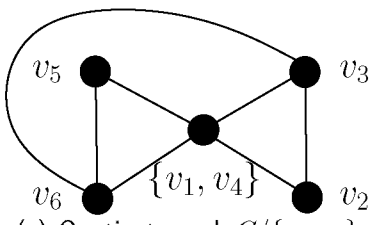
where P and R are defined in Eq. 7.2. Figure 7.5 shows a binary merge collapse that does not perform a count of the merged edges.

	v_1	v_2	v_3	v_4	v_5	v_6
r_1	·	1	1	·	·	1
r_2	1	·	1	·	·	·
r_3	1	1	·	1	·	1
r_4	·	·	1	·	1	·
r_5	·	·	·	1	·	1
r_6	1	·	1	·	1	·

(a) Adjacency matrix A_G

	$\{v_1, v_4\}$	v_2	v_3	v_4	v_5	v_6
$\{r_1, r_4\}$	·	1	1	·	1	1
r_2	1	·	1	·	·	·
r_3	1	1	·	·	·	1
r_4	·	·	·	·	·	·
r_5	1	·	·	·	·	1
r_6	1	·	1	·	1	·

(b) Binary Merge Square $A(G^{/\!/\{v_1,v_4\}})$



(c) Quotient graph $G/\{v_1, v_4\}$

	$\{v_1, v_4\}$	v_2	v_3	v_5	v_6
$\{r_1, r_4\}$	·	1	1	1	1
r_2	1	·	1	·	·
r_3	1	1	·	·	1
r_5	1	·	·	·	1
r_6	1	·	1	1	·

(d) Collapsed Binary Merge Square $A(G^{/\!/\{v_1,v_4\}})$

Figure 7.5: Merging (addition) rows r_1, r_4 of A_G results in a Binary Merge Square

In a Binary Merge Square model a merge results in a simple graph from a simple graph, since it just collapses the multiple edges. The same behaviour can be seen here with the Integer Merge Square model. If some row is merged into the $i-$th row then the a_{ii} elements remain zero due to the merge condition (7.1). A Binary Merge Square is simply the adjacency matrix of the resulting simple graph after a merge. If necessary, A_G can be used to identify the adjacency matrix of the original graph, which contains the clearest representation of the generated problem after a merge. This is quite useful

if an algorithm focuses just the core of the problem.

Merge Algorithms work on Merge Models, performing subsequent merges until the Merge Operation becomes unfeasible. These merges generate a matrix sequence and a corresponding graph sequence. The following definitions identify states of the matrices and graphs during an algorithm run.

Definition 7.1 (Merge Matrix and merge graph) *The Merge Matrix is a general name for a matrix of an integer or Binary Merge Table or square. The merge graph is the corresponding power or quotient graph.*

Definition 7.2 (Initial Merge Matrix and merge graph) *Let G be the graph to be coloured. The initial merge graph is G and the initial Merge Matrix corresponds to the adjacency matrix of G.*

Definition 7.3 (Final Merge Matrix and merge graph) *The final Merge Matrix is that matrix where no more merges are possible. The corresponding merge graph is the final merge graph.*

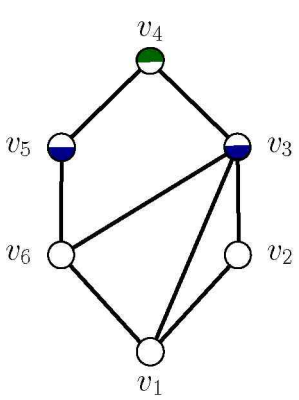
Definition 7.4 (Intermediate merge matrices and merge graphs) *Intermediate merge matrices and merge graphs are those between the initial and final merge matrices and graphs, respectively.*

Generally, when speaking about any of the merge representations we use the term **Merge Matrix** instead of calling them a table or square. Now let M denote a general Merge Matrix in the following. We may associate each row of a Merge Matrix with an appropriate vertex in the corresponding quotient or power graph and designate those vertices as **merge vertices**. The number of non-zero elements the constraints do not decrease by any of the Merge Operations since addition or binary-OR do not decrease entries; that is, non-zero entries remain non-zero. But a zero entry may become non-zero after a merge. This process leads to the saturation of non-zero entries. The main task of the colouring strategies is to control this saturation process and prolong it as much as possible, because the number of rows describe the number of colours used. Hence a prolonged merge sequence leads to fewer rows in the final Merge Matrix. This is a key concept in Merge Algorithms.

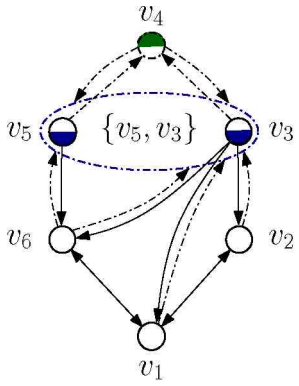
7.2 Sub- and co-structures

Sub-structures Since sequential colouring uses steps where one colour is assigned for each step, the coloured and uncoloured parts of the graph change in a step-by-step fashion. Now it is worth defining the relevant parts of merge matrices separately in the coloured and uncoloured *sub-graphs*. Superscripts ^{col} and ^{unc} stand for partial structures e.g. for sub-merge-matrices M^{col} and M^{unc}, respectively. Figure 7.6 shows these partial structures in the case of Integer Merge Tables, where $M^{col} = \mathbb{T}^{col}$ and

$M^{unc} = \mathbb{T}^{unc}$. Eq. 7.19 reveals more precisely the content of a sub-merge-matrix. The rows in the coloured and uncoloured sub-merge-matrices are referred to coloured and uncoloured rows, respectively.



(a) Partial colouring of G



(b) Relevant power multigraph

	v_1	v_2	v_3	v_4	v_5	v_6
r_4	·	·	1	·	1	·
$\{r_5, r_3\}$	1	1	·	2	·	2

(c) $\mathbb{T}^{unc}$ uncoloured Merge Table

	v_1	v_2	v_3	v_4	v_5	v_6
r_1	·	1	1	·	·	1
r_2	1	·	1	·	·	·
r_6	1	·	1	·	1	·

(d) $\mathbb{T}^{col}$ coloured Merge Table

Figure 7.6: Sub-graphs and sub-merge-matrices of a power multigraph for a partial colouring. The dashed and dotted lines denote the coloured substructure in Figure 7.6(b), while solid lines show the uncoloured structure.

Coloured (and uncoloured) vertices may be characterised by a vector, the characteristic vector, which consist of $1-$s in the appropriate positions and zeros elsewhere.

Definition 7.5 (Characteristic vector of coloured rows) *The characteristic vector of the coloured rows will be denoted by* e^{col}. *The dimension of the vector is equal to the number of rows in the relevant Merge Matrix. Indices of coloured rows define the positions where* e^{col} *have ones, the other entries being all zero.*

Definition 7.6 (Characteristic vector of uncoloured rows) *The characteristic vector of the uncoloured rows is denoted by* e^{unc}. *The dimension of the vector is equal to the number of rows in the relevant Merge Matrix. The indices of uncoloured rows define the positions where* e^{unc} *have ones, the other entries being all zero.*

Figure 7.6 shows a partial colouring where rows $\{\{r_3, r_5\}, r_4\}$ are coloured and $\{r_1, r_2, r_6\}$ are uncoloured. This partial colouring induces the following characteristic vectors:

$$\mathrm{e}^{unc} = (\overset{r_1}{1}, \overset{r_2}{1}, \overset{\{r_5,r_3\}}{0}, \overset{r_4}{0}, \overset{r_6}{1}) \qquad \mathrm{e}^{col} = (\overset{r_1}{0}, \overset{r_2}{0}, \overset{\{r_5,r_3\}}{1}, \overset{r_4}{1}, \overset{r_6}{0}) \tag{7.18}$$

Since r_5 is merged into r_3, position 5 is removed and entry 3 represents the coloured/merged row. The characteristic vectors of rows can be obtained from each other by a simple subtraction; namely $\mathrm{e}^{unc} = \mathrm{e} - \mathrm{e}^{col}$, where e is the vector of all ones. Sub-merge-matrices can be defined like so:

$$M^{col} = \mathrm{Diag}(\mathrm{e}^{col})\ M \qquad M^{unc} = \mathrm{Diag}(\mathrm{e}^{unc})\ M \tag{7.19}$$

where $\mathrm{Diag}(.)$ makes a diagonal matrix where the argument vector is in the main diagonal, and the off-diagonal entries are all zero. In this case M^{unc} and M^{col}-s contain zero rows, they are not collapsed. Figure 7.2(b) shows an example of such an non-collapsed matrix M^{unc}.

Similar characteristic vectors can be defined for vertices of the original graph e_G^{col} and e_G^{unc}, where two $\{0,1\}^n$ vectors characterise the uncoloured and coloured vertices and $n = |V_G|$.

Definition 7.7 (Characteristic vector of coloured vertices) *The characteristic vector of coloured vertices is denoted by* e_G^{col}. *The dimension of the vector is the same as number of vertices in the original graph* G. *Indices of coloured vertices define the positions where* e_G^{col} *have ones, while the other entries are all zero.*

Definition 7.8 (Characteristic vector of uncoloured vertices) *The characteristic vector of uncoloured vertices is denoted by* e_G^{unc}. *The dimension of the vector is the same as the number of vertices in the original graph* G. *Indices of uncoloured rows define the positions where* e_G^{unc} *have ones, while the other entries are all zero.*

Regarding Figure 7.6 the $\{r_3, r_5, r_4\}$ vertices are coloured and the $\{r_1, r_2, r_6\}$ vertices are uncoloured. Hence the characteristic vectors are $\mathrm{e}_G^{col} = (0,0,1,1,1,0)$ and $\mathrm{e}_G^{col} = (1,1,0,0,0,1)$. These vectors are also complementer of each other, since $\mathrm{e}_G^{col} = \mathrm{e} - \mathrm{e}_G^{unc}$. The e^{col} and e^{unc} vectors may be derived from e_G^{col} and e_G^{unc} by simple binary-OR operations (merges) on the relevant indices belonging to merged vertices. In the traditional colouring, the sub-adjacency-matrices are associated with G^{col} and G^{unc} which are sub-graphs of G, they differ from the merge interpretation. Here rows and columns must be removed from the original adjacency matrix, as follows

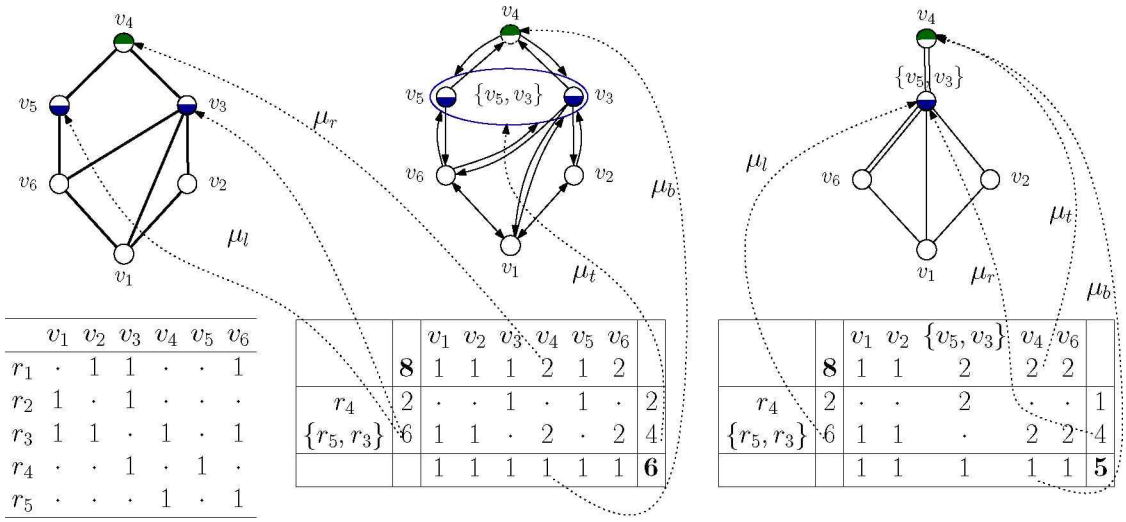
$$A^{col} = (\mathrm{e}_G^{col} \otimes \mathrm{e}_G^{col}) \circ M \qquad A^{unc} = (\mathrm{e}_G^{unc} \otimes \mathrm{e}_G^{unc}) \circ M \tag{7.20}$$

The $(\mathrm{e}_G^{col} \otimes \mathrm{e}_G^{col})$ dyadic product 'masks out' the relevant entries of the adjacency matrix.

First order co-structures are the cells of the representation merge matrices. They define the neighbourhood relation of the merge vertices for the binary and weighted relations for the Integer Models.

Secondary order co-structures or, simply *co-structures*, are the sums of the rows and columns in the representation matrices, respectively. There are four such vectors. We can get the sum of the rows and columns of Binary Merge Matrices from their integer pairs by counting their non-zero elements. Figure 7.7 shows the four co-structures on the four sides of the sub-merge-matrices in the case of Merge Tables and Merge Squares as well. The left hand side contains the sums of the rows of the Integer Models, while the right side contains the sums of the non-zero elements of the rows. This is the same for the columns, where the top vector is the sum of the rows and the bottom is the number of non-zeros. In the case of the Binary Models the left and the right/the top and the bottom co-structures are the same.

The second order structures will be denoted by μ, using $_{t,b,l,r}$ indices as subscripts to refer to the top, bottom, left and right vector, respectively. Figure 7.7 shows sub-merge-matrices for coloured vertices, but co-structures may be defined for the uncoloured part and for the whole Merge Matrix as well. To be consistent, with the previous notations, ^{unc} and ^{col} will be denote the location of the co-structure.



	v_1	v_2	v_3	v_4	v_5	v_6
r_1	·	1	1	·	·	1
r_2	1	·	1	·	·	·
r_3	1	1	·	1	·	1
r_4	·	·	1	·	1	·
r_5	·	·	·	1	·	1

		v_1	v_2	v_3	v_4	v_5	v_6	
	8	1	1	1	2	1	2	
r_4	2	·	·	1	·	1	·	2
$\{r_5, r_3\}$	6	1	1	·	2	·	2	4
		1	1	1	1	1	1	**6**

		v_1	v_2	$\{v_5, v_3\}$	v_4	v_6	
	8	1	1	2	2	2	
r_4	2	·	·	2	·	·	1
$\{r_5, r_3\}$	6	1	1	·	2	2	4
		1	1	1	1	1	**5**

Figure 7.7: The original graph, its sub-Integer Merge Table and then its sub-Integer Merge Square of coloured vertices when colouring is in progress. Here μ_l gives the sum of the degree of the vertices in a colour class, μ_r gives the number of adjacent vertices of a colour class, μ_t gives the number of adjacent coloured vertices, and μ_b gives the number of adjacent colour classes.

Third order co-structures sums the secondary order structures. These may be divided into two parts, like the second order structures, based on the coloured and uncoloured sub-graphs. These structures will be denoted by ζ. In this study, they will be employed in the fitness function of the evolutionary algorithm. The top-left sums ζ_t of the top vector (or the left vector) and the bottom-right sums ζ_b of the bottom vector (or the right vector). These are shown in bold in Figure 7.7. We will also use the ^{unc} or ^{col} notation to distinguish between the parts, while co-structures without any superscript will refer to the whole Merge Matrix.

7.3 Summary

This chapter introduced four different models, called Merge Models, for the graph colouring problem. The models consist of matrix representations and special matrix operations, i.e. Merge Operations. The Merge Operations replace the traditional colour assignments. These models describe graph homomorphisms based on the Quotient and Power methods of Chapter 6. In order to get a colouring algorithm, the algorithm steps must be defined; that is, a sequence of the Merge Operations must be defined.

In the next chapter we will create a general framework for the algorithms based on the Merge Models.

Chapter 8

Merge Frameworks

In Chapter 7 we modelled the graph colouring problem via matrix via matrix representations and operations, starting with the adjacency matrix of the graph. The author introduced general frameworks for graph colouring algorithms supported by Merge Models in [100; 101]. These are generalisations of the traditional colouring schemes of Section 4.2.1. Sequential colouring and independent set methods also fit into these frameworks. This general framework with the new Merge Models supports a common structural analysis of the existing and novel graph colouring methods, as shown by the author in [97; 99; 101; 102].

8.1 The UC and CU Merge Frameworks

There are two options in the case of sequential colouring: either we choose an uncoloured vertex first and then choose a suitable colour for the vertex (UC) or, conversely, we can choose a colour first and then find an appropriate uncoloured vertex for the assignment (CU). These two types may be clearly described by using uncoloured and coloured merge sub-merge-matrices in the UC and CU Merge Frameworks (see Figure 8.1). These frameworks do not provide any selection strategy. However, a combination of particular strategies with a Merge Frameworks results in an algorithm. Consequently, the same strategy with different framework results in different algorithms. The other option for making an algorithm is when different strategies are combined with the same framework. Chapter 10 will give examples for each type. Note that Merge Models work without using colours. Recall that colours serve only to aid understanding; they only indicate whether a row has already been taken into account in the merge process. For this purpose one can use coloured and uncoloured characteristic vectors, as described in Section 7.2. The choose-unc and choose-col functions/strategies are not defined precisely here. They can be replaced by different concrete choice strategies which operate on coloured and uncoloured sub-merge-matrices, respectively. The choose-unc function selects an uncoloured row/vertex, while choose-col selects a coloured row/'colour class' or allocates a new empty row in the coloured sub-merge-matrix. The allocation step introduces a new 'colour'/colour class into the system. In fact, in term of traditional

UC MERGE FRAMEWORK(A adjacency matrix)
1 $M \leftarrow A$
2 **repeat**
3 $u \leftarrow \arg\text{choose-unc}_i\{M_i^{unc}\}$ //Choose an uncoloured row index
4 $c \leftarrow \arg\text{choose-col}_i\{M_i^{col}\}$ //Choose a coloured row index,[a] where $M_{uc} = 0$
5 $M \leftarrow \text{merge}(M, \{u, c\})$ //Merge u and c rows/columns [b]
6 **until** M^{unc} is empty
7 **return** M

CU MERGE FRAMEWORK(A adjacency matrix)
1 $M \leftarrow A$
2 **repeat**
3 $c \leftarrow \arg\text{choose-col}_i\{M_i^{col}\}$ //Choose a coloured row index
4 $u \leftarrow \arg\text{choose-unc}_i\{M_i^{unc}\}$ //Choose an uncoloured row index[c], where $M_{cu} = 0$
5 $M \leftarrow \text{merge}(M, \{u, c\})$ //Merge u and c rows/columns
6 **until** M^{unc} is empty
7 **return** M

[a] $M_{uc} = M_{cu} = 0$ is the merge condition, i.e. there is no edge.
[b] For Merge Squares, columns are also affected in a Merge Operation.
[c] $M_{cu} = M_{uc} = 0$ is the merge condition, i.e. there is no edge.

Figure 8.1: The UC and CU Merge Frameworks

colouring, a merge puts the uncoloured vertex chosen into the selected 'colour class'[1]. Note that a merged row characterises colour classes where a merged row encompasses additional rows of the original adjacency matrix. Substituting colourings by merges, the sequential merge generalises the sequential colouring, where instead of a colour assignment a Merge Operation is performed. It is a generalisation of sequential colouring because on the one hand the traditional colouring schemes can be defined within these frameworks and, on the other hand, traditional schemes can be extended. Section 4.2.1 describes the two traditional sequential colouring schemes. The first is the sequential colour assignment, where vertices get colours in a greedy manner. This may be defined in the UC Merge Framework (see Figure 8.1). An uncoloured row selection by choose-unc means selecting an uncoloured vertex. Then the strategy choose-col can be a greedy coloured row choice. Finally, the colour assignment is equivalent to a merge. The second is the independent set approach, where subsequent independent sets are created in a step-by-step fashion, and each of them is filled with uncoloured vertices until their saturation; that is, no more uncoloured vertices can be encompassed. It may be expressed in the CU Merge Frameworks. The strategy choose-col can create an empty row in the coloured sub-merge-matrix, choose-unc selects a row from the uncoloured sub-merge-matrix, then the rows merged. An independent set represents a colour class; moreover, these colour classes correspond to the rows in a coloured sub-merge-matrix. Thus, an empty row refers to an empty colour-set. In addition, the

[1] A colour class is deemed empty when a new 'colour', a blank row, is created in the coloured sub-merge-matrix.

uncoloured row selected by choose-unc is associated with an uncoloured vertex. Then a merge itself puts the uncoloured vertex into the empty colour class. Later, choose-unc keeps selecting the last row created in the coloured sub-merge-matrix, i.e. the last colour class, until its saturation; that is, no more uncoloured rows can be selected for a merge. These two traditional approaches both apply greedy colour selection for an assignment or greedy vertex filling. The UC and CU Merge Frameworks provide additional possibilities where the greedy choice strategies may be replaced by any other sophisticated one. The task of a choice strategy is to generate choice probabilities for each row of the Merge Matrix. Then, based on the probabilities generated, it selects a row from the uncoloured sub Merge Matrix and another one from the coloured sub Merge Matrix. Depending on the sequence of choices, the algorithm will belong to the UC or the CU Merge Framework. All rows must get a choice probability. Hence, a row choice probability function must be defined to assign probabilities to the rows.

Definition 8.1 (Row choice probability function) *The row choice probability function assigns choice probabilities to each row of the Merge Matrix. A choice probability determines how probable the selection of the two rows is for a merge in the next step of a Merge Algorithm.*

An algorithm in the UC and CU Merge Framework defines two row choice strategies. One is for the rows of the uncoloured sub-merge-matrix, while the other is for the rows of the coloured sub-merge-matrix. These row choice strategies in turn define two row choice probability functions which are the basis for the selection. The probabilities of the row choice probability function may be represented in vector format.

Definition 8.2 (Choice probability vector) *A choice probability vector* $\mathbf{x}$ *contains values of the row choice probability function. The* $\mathbf{x}_i$ *element of the vector represents the choice probability of the* $i-th$ *row for a merge.*

8.2 The CC Merge Frameworks

Notice here that choose-col and choose-unc strategies are compatible in Section 8.1. Both choose a row from the Merge Matrix, but they operate on different subsets of the rows of the matrix. If one defines a choose-col coloured row selection function then one can without any difficulty use it as uncoloured choice strategy choose-unc and vica versa. Now let us exploit this observation and introduce the CC Merge Framework (see Figure 8.2). Since the two choose functions are compatible, use a common one instead of two. Note as well that there is no need to distinguish between coloured and uncoloured sub-merge-matrices; just take only the set of rows and apply the common choose function suitable for all of them. Next, take two different, arbitrary rows from the Merge Matrix which satisfy the merge condition and merge them.
The CC Merge Framework is the most general. Even although it covers the UC and CU Merge Frameworks, it is worth defining them separately so as to have a possibility of categorising the algorithms later Moreover, it is useful in the identification of the

CC MERGE FRAMEWORK(A adjacency matrix)
1 $M \leftarrow A$
2 **repeat**
3 $\{i,j\} \leftarrow \arg\text{choose}_{\{i,j\}}\{M_i, M_j : i \neq j\}$ //Choose two row indices[a], where $M_{ij} = 0$
4 $M \leftarrow \text{merge}(M, \{i,j\})$ //Merge i and j rows/columns
5 **until** M is not mergeable
6 **return** M

[a] $M_{ij} = M_{ji} = 0$ is the merge condition, i.e. there is no edge.

Figure 8.2: The CC Merge Frameworks

traditional schemes [2]. To understand better the behaviour and reason why the CC Merge Framework is so-called, one can represent it as a special independent set scheme. The rows of the Merge Matrix corresponds to colour classes, i.e. independent sets. An algorithm in a CC Merge Framework selects two colour classes/independent sets and creates the union of them, this approach being outlined in Section 3.6.2. This is done by merging two arbitrarily selected rows taken from the whole Merge Matrix. An algorithm terminates when no further merge is possible. Row identifiers of the final Merge Matrix are the colour classes that describe the colouring. The selection of two rows for merging is done by a strategy (Merge Strategy). With Merge Strategy, one may define a choice probability for each pair of vertices.

Definition 8.3 (Row-pair choice probability function) *The row-pair choice probability function assigns choice probabilities to each pair of rows of the Merge Matrix. A choice probability determines how probable the selection of two rows is for a merge in the next step of a Merge Algorithm.*

The row-pair choice probability function is finite function, so the values of the function can be represented in matrix format.

Definition 8.4 (Choice probability matrix) *The choice probability matrix X contains values of the row-pair choice probability function. An x_{ij} element of the matrix determines how probable the selection of row i and j is for a merge in the next step of a Merge Algorithm.*

A Merge Strategy always has an explicit or implicit choice probability matrix for the current problem. The strategy itself can choose the most probable pair of vertices for a merge in the next step or it can apply a probabilistic choice using the values of the choice matrix. The higher the value in a matrix for a vertex pair, the higher the chance for a merge of the vertices. A Merge Strategy always generates this matrix, but sometimes it is hidden, just defined implicitly via a description of the strategy. The key property of a Merge Algorithm is a varying choice probability matrix that converges by progressive merges, to a zero matrix when no more merge is possible. In fact the main

[2] Ususally, traditional schemes fits into the CU and UC Merge Frameworks.

task of the colouring is to find an appropriate choice matrix for each step. Sometimes it is convenient to represent the choice matrix as an $n \times n$ size square matrix like that of adjacency matrix of the original graph. In order to achieve this, one can keep zero rows/columns in a Merge Matrix – i.e. non-collapsed Merge Matrix – to have a size of $n \times n$. An X choice probability matrix contains zeros in the non-zero entry positions of the relevant non-collapsed Merge Matrix because adjacent vertices cannot be merged. Moreover, with x_{ii}-s the diagonal entries are also zeros. Therefore it is reasonable to guarantee this property for each step. E.g. in the case of the Binary Merge Square, the $X \circ \bar{A}$ entrywise product gives the desired result, where $\bar{A}$ is the Binary Merge Square, i.e the adjacency matrix of the complementer quotient graph. The matrix $\bar{A} = J - I - A$ serves as the appropriate Merge Square, where J is the matrix with all one entries and I is the identity matrix. The UC and CU Merge Frameworks divide the problem into uncoloured and coloured parts when an algorithm is running. Merges can be only between two rows which are in different parts; that is, uncoloured and coloured rows can only be merged. Therefore X can only have non-zero values in the relevant cross positions.

8.3 Summary

In this chapter we introduced graph colouring frameworks which generalise the traditional sequential colouring schemes. Each name refers to the applied colouring/merging scheme. Namely, U means an uncoloured vertex and C means a colour class. Hence in a UC Merge Framework an uncoloured vertex is chosen first, then a colour class is associated with it. These frameworks cover and extend the traditional vertex ordering schemes outlined in Section 4.2.1. The CU Merge Framework selects a colour class first, then associates a uncoloured vertex with it. This framework includes the traditional independent set approach of Section 4.2.1 since a colour class is an independent set. In the third framework the CC does not distinguish between colour and uncoloured entities, but takes only colour classes/independent sets then combines them. Note that a single vertex forms an independent sets, hence initially it takes each vertex as one element independent sets and combines them according to a strategy. All of these frameworks are defined in a unified manner using the Merge Model scheme. An algorithm in one of these frameworks applies a subsequent selection of rows of the merge matrices and merges them to achieve a colouring. None of these frameworks has a concrete strategy for the choice of rows for merging. A framework with a concrete row choice strategy forms a particular algorithm.

Chapter 9

Merge Strategies

In chapters 7 and 8 Merge Operations and general Merge Frameworks were defined in order to perform sequential Merge Operations on the original graph and other subsequent merges. A Merge Operation takes two rows/columns of a Merge Matrix and produces a new Merge Matrix if the merge condition allows it. By repeating Merge Operations we will end up with a final Merge Matrix where a Merge Operation is no longer possible. In the case of the Merge Squares, the final Merge Matrix corresponds to a complete graph, while in the case of Merge Tables the final Merge Matrix corresponds to a power graph which is homomorphic with a complete graph. The sequence of the Merge Operations is crucial. It determines the quality of the solution, i.e. the number of colours used in the colouring of the original graph. The number of colours is the same as the number of rows in a Merge Matrix. Hence the main aim is to reduce the number of rows in a Merge Matrix. Each Merge Operation decreases the number of rows by one, until a merge is no longer possible. Therefore the goal is to make as many merges as possible.

This chapter describes various strategies used to generate merge sequences, as described in [94; 96–102] by the author. *Binary Merge Squares or Tables are assumed in the descriptions of the strategies but their integer extensions are also discussed. These strategies proved useful in the theoretical analysis and experimental study. Chapter 10 outlines various algorithms, where these strategies are combined with different Merge Frameworks of Chapter 8. These algorithms were studied by Juhos et al. in [94; 96–102]. The algorithms which apply these strategies outperform several well-known benchmark algorithms from the literature, which were described in Section 4.2.*

In Section 4.2.1 we presented two traditional principles for colouring strategies. The first was the sequential colour assignment scheme, where an uncoloured vertex is chosen first and then a first available colour is assigned to this vertex. Then the second was the maximal independent set approach where the next available colour is taken, then as many vertices as possible are coloured with this colour. The same coloured vertices form maximal independent sets in this case. In both cases the colour choice is greedy and the uncoloured vertex choice is based on some strategy. In Chapter 8,

these approaches were generalised in the UC and CU Merge Frameworks, respectively, where an additional framework called the CC Merge Framework was also introduced. A colour can be interpreted as a colour class that is an independent set. An uncoloured vertex is also an independent set containing a single member. Merge Models associate the colour class with the rows of merge matrices. Each row of a Merge Matrix represents an independent set which can be either a colour class or an uncoloured vertex. The colour assignment operation is implemented as a Merge Operation of the appropriate rows/columns representing a merge of two corresponding independent sets. Hence instead of a colour choice or an uncoloured vertex choice, we may just define a row choice from the set of appropriate Merge Matrix rows. Consequently, the row choice generalises the traditional choice schemes. A row choice may represent either an uncoloured vertex choice or a colour choice in the traditional colouring term, depending on which set of rows of a Merge Matrix forms the basis of the choice. Following traditional colouring schemes, rows of a Merge Matrix will be partitioned into coloured and uncoloured row sets. There are two strategies available to choose a row from the coloured and another from the uncoloured part. The Merge Operation is based on these selections. In the case of Merge Tables it is rows, but in the case of Merge Squares the corresponding columns are merged as well. Depending on the order of the choices from the two row sets, the result will be the UC or the CU Merge Framework. The row selection in the uncoloured and coloured row sets are defined by two row choice strategies. In the traditional schemes one of them is usually a greedy strategy, e.g. take the first row from the coloured row set which is mergeable with a row selected from the uncoloured row set. If the row set is not partitioned, then an algorithm can choose two arbitrary rows for a merge. This approach is defined in the CC Merge Framework. Here, instead of two separated row choices, one row-pair choice is used.

9.1 Row-pair choice strategies

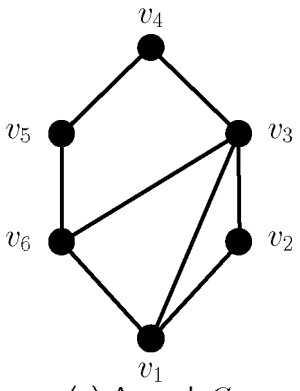
An algorithm in the CC Merge Framework does not make a distinction between the coloured and uncoloured states of the rows of the Merge Matrix. Each row represents an independent set/colour class. Merging the two representation rows results in a union of the independent sets. These algorithms just focus on a sequence of merges of two selected rows of a Merge Matrix. To find such a sequence, a strategy must be a row-pair choice strategy that selects row-pairs successively in order to merge them. This procedure presupposes a row-pair choice probability function (CPF), which assigns a probability value for each row-pair (M_i, M_j) of a Merge Matrix M, in proportion to their chance of being chosen. This is illustrated by a general row-pair CPF:

$$\forall\, M_i, M_j \qquad (M_i, M_j) \rightarrow \begin{cases} x_{ij} & i \neq j \wedge M_{ij} = 0 \\ 0 & i = j \vee M_{ij} \neq 0 \end{cases} \qquad 0 \leq x_{ij} \leq 1 \qquad (9.1)$$

where M_i is the $i-$th row of a Merge Matrix and x_{ij} is the probability of choosing of $i-$th and $j-$th rows for a merge. A choice probability function can be defined by a

choice probability matrix (see Definition 8.4), where the matrix (i, j) element is the x_{ij}. To get reasonable probabilities, the function for non-mergeable row-pairs should be zero, i.e. when $i = j$ and $M_{i,j} \neq 0$. Instead of probabilities x_{ij}, sometimes it is easier to generate $\hat{x}_{ij}$ values which do not necessarily lie in the interval $[0, 1]$, but are correlated with the row-pair CPF. In this case a $[0, 1]$−normalisation provides the corresponding row-pair CPF. A simple $[0, 1]$−normalisation is defined by

$$x_{ij} = \frac{\hat{x}_{ij} - \min_{i,j} \hat{x}_{ij}}{max_{i,j}\hat{x}_{ij} - \min_{i,j} \hat{x}_{ij}} \tag{9.2}$$



(a) A graph G

	v_2	v_6	v_4	v_1	v_5	v_3
r_2	·	·	·	1	·	1
r_6	·	·	·	·	1	1
r_4	·	·	·	·	1	1
r_1	1	1	·	·	·	1
r_5	·	1	1	·	·	·
r_3	1	1	1	1	·	·

(b) Initial Merge Matrix, the adjacency matrix of G

	r_2	r_6	r_4	r_1	r_5	r_3
r_2	·	1	0.6	·	0	·
r_6	1	·	0.6	·	·	·
r_4	0.6	0.6	·	0.3	·	·
r_1	·	·	0.3	·	0.3	·
r_5	0	·	·	0.3	·	0.6
r_3	·	·	·	·	0.6	·

(c) A choice probability matrix

Figure 9.1: A choice probability matrix.

Figure 9.1(c) gives an example for a choice probability matrix (CPM). The columns and rows of the CPM of Figure 9.1(c) correspond to the rows of the Merge Matrix of Figure 9.1(b). Based on the values of the CPM, a strategy can choose two rows for a merge, e.g. the choice probability of the row-pair (r_4, r_2) is 0.6. A choice strategy in a Merge Algorithm calculates the values of the relevant CPM and carries out a deterministic or stochastic selection of two rows for a merge.

9.2 Row choice strategies

The CC Merge Framework does not distinguish between coloured and uncoloured rows. It takes two rows from the Merge Matrix according to a row-pair choice strategy and merges them, then repeats this on the result Merge Matrix. But an algorithm in both UC and CU Merge Frameworks separates the coloured and uncoloured parts. Both choose a row from the uncoloured sub-merge-matrix, i.e. an uncoloured vertex, and another from the coloured sub-merge-matrix which represents a colour class. After making these selections, the two rows are merged, which is the colouring step. Only the sequence of the choices is different. An algorithm in the UC Merge Framework first selects a row from the uncoloured part, then from the coloured part; while an algorithm in the CU Merge Framework changes this order and first selects a coloured row then an uncoloured row. The first selection may have an influence on the second selection.

Note that in traditional colouring schemes (Section 4.2.1) the vertices and colours or colour classes are different objects. In the Merge Model, both correspond to a row

of a Merge Matrix. Hence, a row choice strategy is suitable for choosing an uncoloured vertex or a colour class as well. The only difference is that a row choice strategy must operate on either the coloured sub-merge-matrix or on the uncoloured one. Hence, we may define general row choice strategies. A general row choice strategy can serve as uncoloured or coloured row choice strategy in the UC or CU Merge Frameworks. Different combinations may result in different algorithms. A Merge Algorithm generates two row selections. Hence there must be two, not necessarily different, strategies for these two row choices. To get a row choice strategy, an algorithm must implicitly or explicitly define a *row choice probability function*. This is different from the row-pair choice probability function, which is suitable for the CC Merge Framework. A row CPF assigns selection probability values to single rows instead of row-pairs. These probability values can be represented in vector form, in the row choice probability vector. An algorithm in a UC Merge Framework can create a row choice probability vector in advance, which corresponds with the traditional vertex ordering scheme. In fact, this vector is a row of a suitable choice probability matrix, defined by a 'hidden' row-pair CPF. Usually, a row-pair CPF is implicitly defined through the separated uncoloured and coloured row CPF-s. However, one can combine two row CPF-s to provide a row-pair CPF. The combined function must assign zero probability values for those rows which have the same states, either coloured or uncoloured. The following equation defines a general row-pair CPF for both the UC and CU Merge Frameworks.

$$\forall\, M_i^{s_i}, M_j^{s_j} \quad (M_i^{s_i}, M_j^{s_j}) \rightarrow \begin{cases} x_{ij} & s_i \neq s_j \wedge M_{ij} = 0 \\ 0 & s_i = s_j \vee M_{ij} \neq 0 \end{cases} : \quad \begin{array}{c} 0 \leq x_{ij} \leq 1 \\ s_i, s_j \in \{col, unc\} \end{array} \tag{9.3}$$

Figure 9.2 shows a plot of the choice probability matrix in corresponding with Eq. 9.3, where only uncoloured and coloured rows can be selected for a merge.

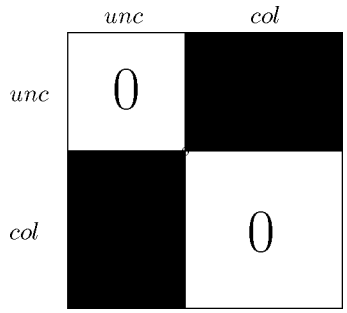


Figure 9.2: The choice probability matrix in UC and CU Merge Frameworks. Only the black parts can have non-zero entries. Here 'col' and 'unc' refer to coloured and uncoloured row indices, respectively.

Combining two row CPF values results in a value pair. However, a comparison of single values may be unambiguous, but a comparison of a value pair is sometimes problematic e.g. take $(3, 1)$ and $(2, 2)$, where $3 > 2$, but $1 < 3$. For all i, let x_i be the choice probability of row M_i generated by a row CPF and construct a flexible choice probability matrix X of the row-pair CPF. Define x_{ij} entries of X according to Eq. 9.5.

$$\hat{x}_{ij} = x_i^{\nu} \cdot x_j^{1-\nu} [M_{ij} = 0] \tag{9.4}$$

$$x_{ij} = \begin{cases} \frac{\max\{\hat{x}_{ij}, \hat{x}_{ji}\}}{\kappa} & i \neq j \\ 0 & i = j \end{cases} \tag{9.5}$$

Eq. 9.4 defines an unnormalised support for the choice. Often only these values form the basis of the decision of a strategy without normalisation. The κ is a normalising constant to get values between zero and one. The merge condition is $(M_{ij} = 0)$. It is expressed by the following Kronecker delta function: $[M_{ij} = 0]$. This function gives one in the case of equality, otherwise results in zero. It can be substituted by $(1 - M_{ij})$ if the x_i-s are non-negatives. The term $(1 - M_{ij})$ is one if rows i and j are mergeable, otherwise it is non-positive, hence only mergeable rows play a role in the selection process. The $\max\{\hat{x}_{ij}, \hat{x}_{ji}\}$ ensures the symmetry of the choice probability matrix. Furthermore, $0 \leq \nu \leq 1$ defines a bias. It favours large values in the combination. In the case of $\nu = \frac{1}{2}$, this strategy favours the selection of rows having large values but not necessarily the largest. In order to favour those rows having the largest CPF value, the $\frac{1}{2}$ bias should be altered. The bias $\nu = 0$ (or $\nu = 1$) result can be utilised as a row CPF, where only one value of the pair is considered. As an example, take two pairs of mergeable rows $(\mathbf{r}_1, \mathbf{r}_2)$ and $(\mathbf{r}_3, \mathbf{r}_4)$ which have the following row CPF values $(2, 2)$ and $(3, 1)$, appropriately. Let $\nu = \frac{1}{2}$, then apart from the normalisation $(\sqrt{3} \cdot \sqrt{1}) < (\sqrt{2} \cdot \sqrt{2})$. The $3^{\nu} > 2$ should be hold to favour the selection of $(\mathbf{r}_1, \mathbf{r}_2)$ pair which have the largest row CPF value 3. Indeed, choose $\nu > \log_3 2$ then $3^{\nu} > 2$ and hence the $(\mathbf{r}_1, \mathbf{r}_2)$ is selected for a merge. Otherwise, when $\nu < \log_3 2$, then the other pair $(\mathbf{r}_3, \mathbf{r}_4)$ is favoured.

9.3 Update mechanism

The relation between the Merge Matrix rows usually changes after a merge. It requires a recalculation of the relevant choice functions. In the UC and CU Merge Frameworks, an uncoloured row is merged into a coloured one. Hence, the affected uncoloured row must be removed, or set to zero, in the uncoloured sub-merge-matrix. Furthermore, the coloured sub-merge-matrix also changes in the affected coloured row. Row choice probability functions have to be updated for the two affected rows. It means that one entry has to be updated in the coloured and another in the uncoloured choice probability vector. The CC Merge Framework needs a row-pair choice probability function. When two rows are merged, the corresponding function values have to be updated. In the choice probability matrix representation of the function values, the appropriate row and corresponding columns have to be updated.

9.4 Extension of non-merge based strategies

A non-merge based choice strategy can be extended using a Merge Model. The extension is based on the transformations of the problem, i.e. merge matrices and associated graphs, induced by Merge Operations. Consecutive Merge Operations generate a Merge

Matrix series. A merge reduces the size of the matrix, producing compact relations where problematic parts may be revealed. Hence these matrices can better characterise the original problem. Intermediate matrices in the matrix sequence may contain more and more information proportional to the number of merges, because intermediate matrices asymptotically approach a final Merge Matrix. An intermediate matrix has the same structure as the initial one in the case of Merge Squares[1]. Consequently, if a strategy can operate on the adjacency matrix, the initial Merge Matrix, then the same strategy can cooperate with the intermediate matrices as well. It introduces a dynamic reconsideration process where previous decisions of a strategy, i.e. CPF-s can be revised by exploiting the additional information contained in the intermediate matrices.

9.4.1 Extended Welsh-Powell ($\infty-$norm) Strategy

Motivation. In Section 4.2.3 we introduced the Welsh-Powell algorithm, where the vertices are ordered in decreasing vertex degree and then greedily coloured. It uses at most $\max_i \min\{d_i + 1, i\}$ colours, where d_i is the degree of the $i-$th position vertex. The degree of a vertex may be calculated by the sum of the relevant row of the adjacency matrix. Hence the vector which contains every degree is the following:

$$\mathbf{d} = A_G\ \mathbf{e} \tag{9.6}$$

where A_G is the adjacency matrix of the original graph and $\mathbf{e}$ is the vector of all one entries. This strategy selects the most constrained uncoloured vertices by edges in a graph. It is represented by the maximum row sum, which is looked for among the rows corresponding to uncoloured vertices. This is the maximum of $\mathbf{d}^{unc}$ of Eq. 9.7, where $\mathbf{d}^{unc} = \mathbf{d} \circ \mathbf{e}_G^{unc}$ and $\mathbf{e}_G^{unc}$ is the characteristic vector of the uncoloured vertices (see Definition 7.8). After colouring a vertex, the search for the maximum row sum proceeds with the rest of the vertices. Hence the maximisation process always just takes the uncoloured degrees (Eq. 9.7), which contains only those rows of the adjacency matrix which correspond to uncoloured vertices.

$$\mathbf{d^{unc}} = A_G\ \mathbf{e}_G^{unc} = \mathbf{d} \circ \mathbf{e}_G^{unc} \tag{9.7}$$

The original Welsh-Power strategy chooses that uncoloured row which has the maximum degree in the original graph; that is, the basis of the choice is d^{unc}. A generalisation of this strategy can be defined by the following merge scheme. The general idea is the same, namely to avoid the possibility that the least constrained vertices collect too many irrelevant vertices, as the original Welsh-Powell method does. The initial Merge Matrix is the adjacency matrix in each Merge Model. After a Merge Operation the Merge Matrix M is transformed into another one, where the number of rows decreases by one, resulting in a reformulated problem graph where the sums of the rows change[2]. Next, we examine the product of M with the vector of all ones e in Eq. 9.8. It provides

[1] Usually an extension is similar for Merge Tables as well.

[2] Without loss of generality, we shall assume that there is no isolated vertex (it has no neighbours).

the relevant sums in a vector. This vector is introduced as a left co-structure of a Merge Matrix in Section 7.2:

$$\mu_l = M\mathbf{e} \tag{9.8}$$

A row in the Merge Models represents a colour class. The sum of the rows have a different meaning in different Merge Models. Some of these are illustrated in Figure 7.7. The Welsh-Powell vertex choice strategy can be defined in the Merge Model using the left co-structure of the Integer or Binary Merge Tables or the Integer Merge Square. However, three combinations out of the twelve[3] result in the original Welsh-Powell algorithm, the others provide extensions of this. Combination of this strategies with different Merge Models and Merge Frameworks results in different colouring algorithms. They are the Extended Welsh-Powell strategies, which were introduced by *the author* in [97].

Definition 9.1 (Extended Welsh-Powell strategies) *Extended Welsh-Powell strategies are those strategies which are defined by a Merge Model in a Merge Framework using the maximum row sum choice strategy.*

The Welsh-Powell strategy can be defined by uncoloured row choices in several Merge Models. However, an Extended Welsh-Powell strategy can apply the maximum row sum strategy for coloured rows as well. When uncoloured and coloured rows are chosen separately, then the suitable Merge Framework for these type of algorithm are the UC and CU Merge Frameworks. The row choices are always supported by a row choice probability function. Here it was based on the row sums.

The row choice probability function for the UC and CU Merge Frameworks. Both UC and CU Merge Frameworks choose a row from the uncoloured sub-merge-matrix and another from the coloured. Only the sequence of the choices differs. The row choice probability function for the uncoloured sub-merge-matrix is defined in Eq. 9.9. Its counterpart for the coloured sub-merge-matrix is defined in Eq. 9.10. The values of the functions as a sequence can be written as vectors, the choice probability vectors $\mathbf{x}^{unc}$ and $\mathbf{x}^{col}$, respectively.

$$\mathbf{x}_i^{unc} = \frac{\langle M_i^{unc}, \mathbf{e}\rangle}{\kappa} \tag{9.9}$$

$$\mathbf{x}_i^{col} = \frac{\langle M_i^{col}, \mathbf{e}\rangle}{\kappa} \tag{9.10}$$

where M_i^{unc} and M_i^{col} are the appropriate uncoloured and coloured row vectors, respectively and $\mathbf{e}$ is the vector of all one entries. of The $i-$the component of the choice probability vector $\mathbf{x}_i^{unc}$ of $\mathbf{x}_i^{col}$ describes the chance of a selection of the $i-$th uncoloured or coloured row. These choice vectors may be applied separately or they can be combined together with other row choice strategies. The κ must be a reasonable constant

[3]Four types of Merge Models and three variants of the Merge Frameworks.

to normalise the values to get probabilities, e.g. the maximum of the possible row sums. These choice probability vector supports the row choices, which can be either deterministic of stochastic. The deterministic choice strategy for both uncoloured or coloured rows is defined in Eq. 9.11. The stochastic choices is based on a random value generation by some probability distribution. The choice will be the index of that component of the choice probability vector which has the nearest value to generated a random value $0 \leq rnd \leq 1$, as stated in Eq. 9.12.

$$\arg\max_i \mathbf{x}_i^s \qquad s \in \{col, unc\} \tag{9.11}$$

$$\arg\min_i \{|\mathbf{x}_i^s - rnd|\} \qquad s \in \{col, unc\} \tag{9.12}$$

where x_i^s is the appropriate uncoloured or coloured part of the vector. The minimisation problem of Eq. 9.12 results in the index of the closest $\mathbf{x}_i^s$ element to the generated value of rnd. The maximum row sum strategy may be interpreted via the induced ∞-norm of matrices. The ∞-norm provides the maximum row sum:

$$\|M^{unc}\|_\infty = \max_i \{\langle M_i^{unc}, \mathbf{e}\rangle\} \tag{9.13}$$

In order to define a row-pair choice probability function for the CC Merge Framework we will follow the construction of Section 9.2.

The row-pair choice probability function for the CC Merge Framework. Uncoloured and coloured row choices are needed within the UC an CU Merge Frameworks. The CC Merge Framework does not make any distinction between coloured and uncoloured states, but the maximum row sum strategy can be exploited in this framework as well. In the CC Merge Framework, choose two rows for a merge; if they are mergeable and they represent the maximum row sum combination. Since 'combination' is not an exact term here, make use of Definition 8.4 and introduce a choice probability matrix where row sums can be 'combined'. An (i, j) entry of the choice probability matrix represents the chance that the i and j rows will be involved for a merge. Hence it defines the row-pair choice probability function:

$$\hat{x}_{ij} = \langle M_i, \mathbf{e}\rangle^\nu \cdot \langle M_j, \mathbf{e}\rangle^{1-\nu} [M_{ij} = 0] \tag{9.14}$$

$$x_{ij} = \begin{cases} \frac{\max\{\hat{x}_{ij}, \hat{x}_{ji}\}}{\kappa} & i \neq j \\ 0 & i = j \end{cases} \tag{9.15}$$

where κ is a normalising constant needed to get values less than one. Moreover, the x_{ij}-s are all non-negative numbers. The κ can be the maximum of the x_{ij} entries or the sum of the entries $|M|$. In the case of $\kappa = |M|$, the sum of the entries of the choice probability matrix will be one. The sum of the entries of the X choice matrix can be also a good option for normalising constant. The Kronecker delta $[M_{ij} = 0]$ represents the merge condition when the merge condition is not satisfied, i.e. $M_{ij} \neq 0$,

the function give 0, otherwise 1. In the case of Binary Merge Matrices the Kronecker delta function can be substituted in Eq. 9.14, thus

$$\hat{x}_{ij} = \langle M_i, \mathbf{e} \rangle^{\nu} \cdot \langle M_j, \mathbf{e} \rangle^{1-\nu} (1 - M_{ij}) \tag{9.16}$$

In the case of Binary Merge Squares, an equivalent choice strategy can be defined by

$$X = (A\mathbf{e})(A\mathbf{e})^T \circ \bar{A} \tag{9.17}$$

where A is a Binary Merge Square, i.e. the adjacency matrix of an appropriate quotient graph, which is a simple graph. Here $\bar{A}$ is the adjacency matrix of the complementer graph of the quotient graph. It has zeros in the edge positions of A and ones elsewhere, except along the main diagonal which has zeros too. An entrywise product with $\bar{A}$ is a suitable choice because it retains only those array positions where $A_{ij} \neq 0$, while the other array positions will have zero values. Figure 9.3(b) shows a typical choice probability matrix. The Integer Merge Table of Figure 9.3(a) comes from Figure 7.6 with $\nu = 0.5$. The matrix has positive-valued elements only in the possible merge positions.

	v_1	v_2	v_3	v_4	v_5	v_6
r_1	·	1	1	**0**	·	1
r_2	1	·	1	0	·	0
$\{r_3, r_5\}$	1	1	·	2	·	2
r_4	**0**	0	1	·	1	0
r_5	·	·	·	·	·	·
r_6	1	0	1	0	1	·

(a) Integer Merge Table. The 0-s are the possible merge positions.

	r_1	r_2	$\{r_3, r_5\}$	r_4	r_5	r_6
r_1	·	·	·	**1**	·	·
r_2	·	·	·	0.8	·	1
$\{r_3, r_5\}$	·	·	·	·	·	·
r_4	**1**	0.8	·	·	·	1
r_5	·	·	·	·	·	·
r_6	·	1	·	1	·	·

(b) Choice probability matrix. A greedy choice is given in bold.

Figure 9.3: The $\infty-$norm choice probability matrix.

For Merge Squares the induced $\infty-$norm is equivalent to the induced $1-$norm, but for the Merge Tables these norms give different results, since the columns and the rows belongs to different objects. Columns refers to the vertices in the original graph, but the rows correspond to colour classes. A Binary Merge Table describes the relation between the vertices and the colour classes. The $1-$norm provides the maximum column sum, while the $\infty-$norm provides the maximum row sums. A column sum in the coloured sub-merge-matrix gives the number of neighbour colours of a vertex as depicted in Figure 7.7, i.e. the colour saturation degree (Def. 4.3). Therefore the DSatur algorithm in Section 4.2.5 can be defined by this value as well, while the $\infty-$norm belongs to the number of neighbours; and hence, this value helps the Welsh-Powell algorithm.

Improvement. The Extended Welsh-Powell methods apply merges. The number of rows decrease by merges. The $\max\min\{d_i, i\}$ is translated into $\max\min\{\langle M_i\mathbf{e}\rangle, i\}$,

where M_ie is the sum of the i-th row of the Merge Matrix M and i is the position of the row by the choice probability vector. That rows are ordered by their sums. Thanks to the decrease in the number of rows, the i-th position decreases, which can bring improvement in the bound. Moreover, in the case of Binary Merge Squares the sum of the rows must be decrease after a certain number of merges; because the number of columns also decreases after each merge. A final Binary Merge Square of a $k-$colouring has k number of rows and columns and the common degree is $k-1$. We may suppose that the minimal degree δ of the vertices of the original graph satisfies $\delta \geq k-1$, thanks to the low degree reduction technique (see Section 3.4). Therefore, after certain merge steps the sum of the rows must approach[4] $k-1$, leading to a decrease in the upper bound.

9.4.2 Extended Hajnal Strategy

Motivation. In Section 4.2.4 we outlined the Hajnal algorithm, which utilises vertex ordering according to the decreasing values of the components of the principal eigenvector of the adjacency matrix of the original graph A_G. Then a greedy colour assignment to the vertices assures that an upper bound of the number of colours is used. This bound is the principal (the largest) eigenvalue λ_{max}. This strategy was extended by *the author* in the UC Merge Framework, taking advantage of the Binary Merge Squares, because they are the adjacency matrices of the appropriate quotient graphs. After a merge, we employ the strategy for the result Merge Square, and continue in the same fashion as long as a merge is still possible. The Gerschgorin disk theorem [64; 149] and the bound $\lambda_{max} \leq \Delta$ give the same upper bound for the principal eigenvalue, which is the maximum degree in a graph. In this case it is a quotient graph, which is also a simple graph like the original graph. Hajnal strategies improve the upper bound better than the Welsh-Powell 4.2.3 strategy does. This bound has been improved still further by *the author* in the Merge Models.

Improvement. In the Binary Merge Square model, consecutive merges contract the original graph G to quotient graphs until the complete graph K_k is reached, where $k \leq |V_G|$ and no further merges are possible. K_k is a $k-$regular graph, hence its principal eigenvalue $\lambda_{max}(K_k) = k-1$ (see [46]). Moreover, we may suppose that the minimal degree satisfies the constraint $k-1 \leq \delta_G$. Otherwise, apply a low degree vertex removal[5] (see 3.4). According to [45], the following holds: $\delta_G \leq \lambda_{max}(G)$, from which we conclude that $\lambda_{max}(K_k) = k-1 \leq \delta_G \leq \lambda_{max}(G)$. The principal eigenvalue of a final Binary Merge Square must be not greater than the principal eigenvalue of the original adjacency matrix. Thus

$$\lambda_{max}(K_k) \leq \lambda_{max}(G) \tag{9.18}$$

The difference between the value of the left hand and right hand side of Eq. 9.18 can

[4] $k-1$ is the sum of each row in a final Binary Merge Square.

[5] A vertex removal does not increase λ_{max} [45].

be large [6]. So that the application of the Hajnal strategy with the Binary Merge Square model can significantly reduce the upper bound of this strategy in an intermediate quotient graph, which resides between G and K_k. The experimental results in Section 11.3 show this improvement in practice. To illustrate the relationship between the vertices and the components of an eigenvector, take the eigenvalue-equation $A\hat{\mathbf{x}} = \lambda\hat{\mathbf{x}}$. It is reasonable to denote the eigenvector by 'hat' ($\hat{\mathbf{x}}$), because it is the basis of the row choice probability vector, but it should be normalised by its largest element. Now let us consider the fourth component $\hat{\mathbf{x}}_4$ of an eigenvector $\hat{\mathbf{x}}$ and examine the eigenvalue-equation; namely $A_4\hat{\mathbf{x}} = \lambda\hat{\mathbf{x}}_4$. The A_4 defines the neighbours of the v_4 vertices and $A_4\hat{\mathbf{x}}$ sums the eigenvector components related to the neighbours of v_4, enlarging the $\hat{\mathbf{x}}_4$ by λ. Figure 9.4 shows how the eigenvector components are related to each other in a graph.

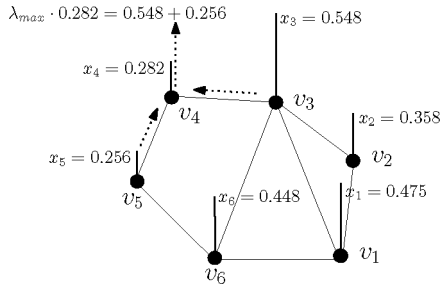


Figure 9.4: The principal eigenvector components correspond to graph vertices. Here to the solid thick lines denote the eigenvector components, while the thin lines are the edges. The arrows graphically represent the relationship $\lambda_{max}\hat{\mathbf{x}}_4 = A_4\hat{\mathbf{x}}_{max} = \hat{\mathbf{x}}_3 + \hat{\mathbf{x}}_5$.

Take the graph G in Figure 9.4 as an example and sort the vertex identifiers of G in decreasing order, according to the components of the principal eigenvector of G.

$$\begin{array}{rcccccc} & \mathring{v}_3 & v_1 & v_6 & v_2 & v_4 & \mathring{v}_5 \\ \hat{\mathbf{x}}_{max} = & (0.548 & 0.475 & 0.448 & 0.358 & 0.282 & 0.256) \end{array} \tag{9.19}$$

The vertex v_3 gets the highest component value in the sorting, but the only mergeable vertices are v_5 and v_3, each being marked by a ring in Eq. 9.19. After merging v_3 and v_5 and again calculating the principal eigenvector for the resulting quotient graph, we get the following:

$$\begin{array}{rcccccc} & \{v_3, v_5\} & \mathring{v}_1 & v_2 & v_6 & \mathring{v}_4 \\ \hat{\mathbf{x}}_{max} = & (0.582 & 0.523 & 0.412 & 0.217 & 0.256) \end{array} \tag{9.20}$$

Figure 9.5 shows the resulting graph of the merge $\{v_3, v_5\}$, where the λ_{max} value is reduced from 2.853 to 2.685. Hence, the upper bound is decreased. The $\{v_3, v_5\}$ vertex is adjacent to all the other vertices, hence v_1 must be selected next. Its only non-neighbour vertex is v_4. So both are designated for the next merge. As it stands this strategy does not work with Merge Tables. However, there is a way to extend it.

[6] The difference between the δ_G and $\bar{d}_G$ values may be large and $\lambda_{max}(K_k) \leq \delta_G \leq \bar{d}_G \leq \lambda_{max}(G)$ (see [45]).

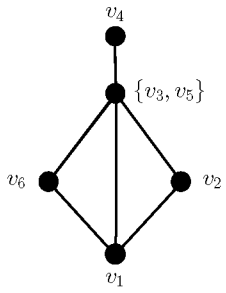


Figure 9.5: A quotient graph produced by the Extended Hajnal strategy.

Take the principal, the largest singular value $\sigma_{max}(M)$ of a Merge Matrix M and the corresponding left singular vector called the principal left singular vector. In the case of a symmetric matrix (like a Merge Square), singular values are eigenvalues and singular vectors are eigenvectors, respectively. Merge Tables are not square matrices[7], but the principal left singular vector can be used to determine an order of the rows, much like that for Merge Squares.

Definition 9.2 (Extended Hajnal strategies) *Extended Hajnal strategies are those strategies which are defined by a Merge Model in a Merge Framework using the first left singular vector choice strategy. The left singular vector components are associated with the vertices. A higher value means a higher chance for selection of the row in the next merge.*

Vertex ordering strategies like the Hajnal strategies correspond to either the UC or CU Merge Framework. There must be two row choice probability functions defined in both frameworks based on possible row choices.

The row choice probability function for the UC and CU Merge Frameworks. The choice probability vector for choosing either from the coloured or uncoloured sub-merge-matrices can be defined by the relevant part of the following choice probability vector like so

$$\mathbf{x} = \frac{\hat{\mathbf{x}}}{\kappa} \quad , \quad \kappa = max_i\{\hat{\mathbf{x}}_i\} \tag{9.21}$$

Similar to the equations 9.11 and 9.12 a deterministic or stochastic choice can be defined.

$$\arg\max_i \mathbf{x}_i^s \qquad s \in \{col, unc\} \tag{9.22}$$

$$\arg\min_i \{|\mathbf{x}_i^s - rnd\} \qquad s \in \{col, unc\} \tag{9.23}$$

The row-pair choice probability function can be defined based on the row choice probability function described in Section 9.2.

The row-pair choice probability function for the UC and CU Merge Frameworks. If $\hat{\mathbf{x}}$ is the left principal singular vector, then the components of the choice

[7] Except for the zeroth Merge Table, which is the adjacency matrix of the original graph.

probability matrix x_{ij} are defined for the Extended Hajnal strategy, namely

$$x_{ij} = \begin{cases} \frac{\hat{\mathbf{x}}_i^{\nu} \cdot \hat{\mathbf{x}}_j^{1-\nu}}{\kappa} (1 - M_{ij}) & i \neq j \\ 0 & i = j \end{cases} \tag{9.24}$$

Note that every $\hat{\mathbf{x}}_i$ is non-negative due to the principal eigenvector Perron-Frobenius property [140].

9.5 Spectral Norm – 2−norm Strategy

Motivation. The Hajnal heuristic (see Section 4.2.4) provide an upper bound for the number of colours used in a colouring. The bound is equal to the principal eigenvalue, which defines the spectral norm of the adjacency matrix. Unfortunately, the principal eigenvalue may be far away from the chromatic number, as mentioned in Section 3.4. Therefore, the Extended Hajnal strategy (Section 9.4.2) tries to exploit the fact that merges can bring a decrease in the eigenvalue in the case of Binary Merge Squares. First, let the Merge Matrix be a Binary Merge Squares. Then applying the Hajnal strategy after a merge, the upper bound should decrease until the process gets to the final Merge Matrix. A final Merge Matrix represents a complete graph K_k on k−vertices. The principal eigenvalue is $k-1$ in this case because K_k is $(k-1)$−regular, as mentioned in Section 9.4.2. The chromatic number is the smallest among the possible k-s, i.e. K_χ is the smallest complete graph which can be produced by a merge sequence. Therefore, $\chi - 1$ is the smallest principal eigenvalue which can be achieved. Exploiting this observation, *the author* introduced a steepest descent spectral norm strategy in [101]. The spectral norm strategy selects two rows from the Merge Matrix which can minimise the spectral norm of the resulting Merge Matrix. Figure 9.14(a) shows how the spectral norm evolves in the intermediate Merge Matrix cases. There are random colourings and an optimal colouring of a 20−chromatic graph. Moreover, the figure contains the values associated with the spectral norm minimisation strategy. Here 24 colours are used in the colouring process by the strategy, while random colouring uses over 36 colours. To get a k−colouring, $|V_G| - k$ merge steps are necessary. Therefore, the curves of Figure 9.14(a) never reach $|V_G| = 200$, but they were extended to get a better insight into how the final colouring is realised. Here, not just the Binary Merge Squares approach can benefit from this strategy. The spectral norm is defined by the principal singular value σ_{max}, hence it can be applied to non-square matrices as well. Thanks to this fact, the strategy works with Binary Merge Tables as well. Furthermore, the spectral norm is equivalent to the induced 2−norm [69; 89], thus

$$\|M\|_2 = \sigma_{max}(M) \tag{9.25}$$

The row-pair choice probability function for the CC Merge Frameworks. Using Eq. 9.25, define the row-pair choice probability function of the steepest descent

spectral norm strategy by $\|\cdot\|_2$ like so

$$x_{ij} = \begin{cases} \frac{\kappa}{\|M_{/ij}\|_2}(1 - M_{ij}) & i \neq j \\ 0 & i = j \end{cases} \tag{9.26}$$

Here $M_{/ij}$ represents the Merge Matrix, which is derived from the Merge Matrix M by merging the $i-$th and $j-$th rows, while M_{ij} is the $(i,j)-$th component of the matrix. The constant κ is a normalisation constant that prevents the values from getting too small. Note that for the Integer Merge Matrices the strategy must be the opposite. Since they keep all the original edges, the values of the entries increase when the size of the matrix decreases. Because $\frac{1}{\sqrt{n}}\|M\|_\infty \leq \|M\|_2$ (see [69]), the increasing value of the spectral norm will define the strategy.

The row choice probability function for the UC and CU Merge Frameworks. The spectral norm strategy like other row-pair choice strategies can work in the UC and CU Merge Frameworks as a second row choice strategy. When an arbitrary strategy selects a row from the uncoloured sub-merge-matrix, it designates a row vector from the choice probability matrix which satisfies Eq. 9.26. The second row selection from the coloured sub-merge-matrix can be done by the steepest descent spectral norm choice along the designated row vector of the choice probability matrix.

The strategy must generate trial matrices like $M_{/ij}$ in order to get the appropriate choice probability values which constitute the choice probability matrix. However, the strategy needs to make as many calculations as the number of mergeable row-pairs. After a merge the number of mergeable elements is reduced, hence the generation of the choice matrix speeds up. Nevertheless, the calculation of the principal eigenvalue can be done efficiently [69]. In practice, the calculation is problematic with large graphs. Fortunately, there are suitable techniques available to get a good approximation of the value for the spectral norm [123]. This approximation helps speed up the calculation of the choice probability values, because the update technique of Section 9.3 can be utilised.

9.6 Spectral norm approximation strategies

Motivation. The spectral norm strategy must first make several trial merges. With the resulting trial merge matrices, the spectral norm strategy makes spectral norm calculations to create a suitable row-pair choice probability function. Calculating the spectral norm of the $M_{/ij}$ is computationally expensive, but Merikoski and Kumar once introduced an efficient spectral norm approximation in [123]. Based on their results, *the author* [101] adapted his spectral norm strategy to an approximated spectral norm strategy, which produced effective calculations of the choice probabilities, where the approximation can be given by the entries of the Merge Matrix. Let $M = A$ be a

Binary Merge Square then

$$\|A_{/ij}\|_2 \approx \sqrt{\frac{\sum_{r=1}^{l} \langle (A_{/ij})_r, \mathrm{e} \rangle^2}{l}} \tag{9.27}$$

where l is the number of rows of the Merge Matrix $A_{/ij}$. $\langle (A_{/ij})_r, \mathrm{e} \rangle^2$ is the square of the r-th row sum, which may be replaced by $\langle (A_{/ij})_r, (A_{/ij})_r \rangle^2$ in the case of Binary Merge Matrices. Since Merge Tables are non-symmetric matrices[8], the symmetric $T_{/ij}T_{/ij}^T$ must be applied[9] in Eq. 9.27 instead of $T_{/ij}$. This leads to an approximation of the square of the spectral norm of $M_{/ij}$. Note that Eq. 9.27 must be applied for the Merge Matrices produced by trial merges in order to get choice probability values (see Eq. 9.26), but a direct calculation of these values is also possible, as described below.

The row-pair choice probability function for the CC Merge Framework. The spectral norm strategy makes as many trial merges as the number of choice probabilities required for the decision. *The author*, improving on earlier results, introduced a choice probability calculation without applying any trial merges. The choice probabilities are calculated directly, using just the Merge Matrix entries. Owing to this, this strategy can exploit the update mechanism of Section 9.3, which is not present in the original spectral norm strategy (Section 9.5).

In the case of an Integer Merge Matrix M a direct calculation of the x_{ij} choice probability values for the i-th and j-th rows is performed using the formula

$$\|M_{/ij}\|_2 \approx \sqrt{\frac{\sum_{r \neq i, r \neq j} \langle M_r, \mathrm{e} \rangle^2 + (\langle M_i, \mathrm{e} \rangle + \langle M_j, \mathrm{e} \rangle)^2}{l}} \tag{9.28}$$

where l is the number of rows of the 'trial' Merge Matrix $M_{/ij}$. After merging rows i and j of M, the row sums do not change in the resulting $M_{/ij}$, except for the i-th and j-th rows: M_i and M_j, which will change as follows: $(M_{/ij})_i = M_i + M_j$ and $(M_{/ij})_j = 0$. Therefore, the $i-$th row sum will be $\langle M_i, \mathrm{e} \rangle + \langle M_j, \mathrm{e} \rangle$, while the j-th row sum will be zero. With a Binary Merge Matrix M, this situation is a bit more complicated because the piecewise OR operation results in 1-s for the common 1 values. That is, if $(M_{/ij})_{ir} = 1$ and $(M_{/ij})_{jr} = 1$ then they result in the merged row $(M_{/ij})_{ir} \vee (M_{/ij})_{jr} = 1$. Figure 9.6 shows a typical example for the case of a Binary Merge Table and Square as well. Let $\mathcal{I}$ be an index set, the set of the common one positions of the rows M_i and M_j. In this example it is $\mathcal{I} = \{4, 6\}$ (see Figure 9.6). Let $M = A$ be a Binary Merge Square, the changes in the row sums after the row A_i

[8] Except for the initial Merge Table, which is the adjacency matrix.

[9] Actually, this form may be suitable for Merge Squares as well, but causes extra computation effort, hence in this case it is not recommended.

is merged into A_j can be summarised as follows

$$\begin{aligned}
\big\langle (A_{/ij})_i, \mathbf{e} \big\rangle &= \langle A_i, \mathbf{e} \rangle + \langle A_j, \mathbf{e} \rangle - \langle A_i, A_j \rangle \\
\big\langle (A_{/ij})_j, \mathbf{e} \big\rangle &= 0 \\
\big\langle (A_{/ij})_r, \mathbf{e} \big\rangle &= \langle A_r, \mathbf{e} \rangle - 1 \quad r \in \mathcal{I} \\
\big\langle (A_{/ij})_r, \mathbf{e} \big\rangle &= \langle A_r, \mathbf{e} \rangle \quad r \notin \mathcal{I} \cup \{i, j\}
\end{aligned} \tag{9.29}$$

Here $A_{/ij}$ denotes the Merge Matrix after merging the i-th and j-th rows. The i-th row and j-th row must be added, hence their sums are added, but the sum of the common row positions $\langle A_i, A_j \rangle$ must be subtracted, due to the OR operation. Besides these changes, based on a merge of the two appropriate column, the row sums are changed by -1 in the $\mathcal{I}$ positions. Note that the merge condition $A_{ij} = A_{ji} = 0$ ensures that the index set $\mathcal{I}$ will never contain i and j indices.

$$\begin{array}{cccccccc}
 & & & 4. & & 6. & & \\
 & \cdot & 1 & 1 & \cdot & \cdot & 1 \\
 & 1 & \cdot & 1 & \cdot & \cdot & \cdot \\
* & 1 & 1 & \cdot & \mathbf{1} & \cdot & \mathbf{1} \\
 & \cdot & \cdot & 1 & \cdot & 1 & \cdot \\
* & \cdot & \cdot & \cdot & \mathbf{1} & \cdot & \mathbf{1} \\
 & 1 & \cdot & 1 & \cdot & 1 & \cdot
\end{array}
\Longrightarrow
\begin{array}{cccccccc}
 & & & & * & & * & \\
 & \cdot & 1 & 1 & \cdot & \cdot & 1 \\
 & 1 & \cdot & 1 & \cdot & \cdot & \cdot \\
* & 1 & 1 & \cdot & \mathbf{1} & \cdot & \mathbf{1} \\
 & \cdot & \cdot & 1 & \cdot & 1 & \cdot \\
* & \cdot & \cdot & \cdot & \mathbf{0} & \cdot & \mathbf{0} \\
 & 1 & \cdot & 1 & \cdot & 1 & \cdot
\end{array}
\Longrightarrow
\begin{array}{cccccc}
 & \cdot & 1 & 1 & \cdot & 1 \\
 & 1 & \cdot & 1 & \cdot & \cdot \\
 & 1 & 1 & \cdot & \mathbf{1} & \cdot & \mathbf{1} \\
 & \cdot & \mathbf{1} & 1 & \cdot & \mathbf{0} & \cdot \\
 & \cdot & \cdot & \cdot & \mathbf{0} & \cdot & \mathbf{0} \\
 & 1 & \mathbf{1} & 1 & \cdot & \mathbf{0} & \cdot
\end{array}$$

Figure 9.6: The $*$ rows and columns are assigned for a merges. The left matrix is the initial Merge Matrix, namely the adjacency matrix. The middle is a Binary Merge Table after merging $*$ rows. The right is a Binary Merge Square, after merging $*$ columns. Common ones and their amendments are shown in bold. The common one positions are $\mathcal{I} = \{4, 6\}$.

For a Binary Merge Table T the approximation may be performed using the symmetric $T_{/ij}T_{/ij}^T$ matrices, where the results of the approximations are the squares of the spectral norm values. Nevertheless, an alternative efficient strategy can be the application of Eq. 9.27 for T only. Since just the rows are merged in the case of Merge Tables, the columns remain unaffected (see Figure 9.6). The changes in the row sums after merging the i-th and j-th rows may be represented by the following

$$\begin{aligned}
\big\langle (T_{/ij})_i, \mathbf{e} \big\rangle &= \langle T_i, \mathbf{e} \rangle + \langle T_j, \mathbf{e} \rangle - \langle T_i, T_j \rangle \\
\big\langle (T_{/ij})_j, \mathbf{e} \big\rangle &= 0 \\
\big\langle (T_{/ij})_r, \mathbf{e} \big\rangle &= \langle T_r, \mathbf{e} \rangle \quad r \notin \{i, j\}
\end{aligned} \tag{9.30}$$

In the case of Integer Merge Tables, the component $-\langle T_i, T_j \rangle$ must be removed from the $\big\langle (T_{/ij})_i, \mathbf{e} \big\rangle$ calculation, because the Merge Operation is the addition operation, while the others sums remain the same.

Remark. The row choice probability function values are generated in the same way as that described in Section 9.5. Kumar and Merikoski in [123] offer more sophisticated approximations for the spectral norm which can be utilised as well. Notice that the

components of this strategy contain row sums like those in the Welsh-Powell method, but here all the row sums are taken into account in a row-pair selection. Moreover these row sums correspond to a Merge Matrix which is derived from the actual Merge Matrix by a merge. However, there may be rows which remain unchanged. Also notice that in the binary cases, the dot product of the two rows also have an influence on the selection. The dot product in the strategy takes the relation of the rows into account as well as their individual properties, like a row sum. The bigger the dot product value, the greater the decrease in the row sum. The following strategy focuses on an analysis of the observed relation between the two rows.

9.7 Dot Product (entrywise norm) Strategy

Motivation. As we saw earlier, the number of rows in a final Merge Matrix defines the number of colour classes used in the colouring problem. Hence the aim is to reduce the number of the rows as much as possible in order to have as few colours as possible. The non-zero elements, the edges can only prevent the further reduction of the rows. Since merges result in changes in the number of non-zero elements, this strategy keeps the number of edges as low as possible in the intermediate Merge Matrix stage. To achieve this goal, two rows must be chosen whose merge reduces the highest number of non-zero elements in the Merge Matrix. Figure 9.7 shows an example.

$$
\begin{array}{cc|cccccc}
 & & \cdot & 1 & 1 & \cdot & \cdot & 1 \\
 & & 1 & \cdot & 1 & \cdot & \cdot & \cdot \\
* & & 1 & 1 & \cdot & \mathbf{1} & \cdot & \mathbf{1} \\
 & & \cdot & \cdot & 1 & \cdot & 1 & \cdot \\
* & & \cdot & \cdot & \cdot & \mathbf{1} & \cdot & \mathbf{1} \\
 & & 1 & \cdot & 1 & \cdot & 1 & \cdot
\end{array}
\quad \Longrightarrow \quad
\begin{array}{cccccc}
\cdot & 1 & 1 & \cdot & \cdot & 1 \\
1 & \cdot & 1 & \cdot & \cdot & \cdot \\
1 & 1 & \cdot & \mathbf{1} & \cdot & \mathbf{1} \\
\cdot & \cdot & 1 & \cdot & 1 & \cdot \\
\cdot & \cdot & \cdot & \mathbf{0} & \cdot & \mathbf{0} \\
1 & \cdot & 1 & \cdot & 1 & \cdot
\end{array}
$$

Figure 9.7: A merge of the $*$ rows causes the greatest reduction in the number of non-zero elements. The figure shows this reduction in a Binary Merge Table.

It leads to the recognition that those rows which have the maximal number of common non-zero elements should be chosen for a merge. In the adjacency matrix, common non-zero entries in two rows mean common neighbours of the corresponding vertices. This strategy was introduced in [97] by *the author*. The strategy with various Merge Frameworks was analysed by Juhos et al. in [98–101]. The results of the analysis will be presented in Chapter 10.

The row-pair choice probability function for the CC Merge Framework. A description of this choice strategy by merge matrices will help keep the definition simple. First let M be a Binary Merge Table or Merge Square. The common non-zeros of the row pairs are provided by the Gram Matrix of M, i.e. MM^T, which consists of the dot products of the rows. Although the Gram Matrix contains essential information, further processing is required to get the mergeable positions because not all positions refer to

mergeable rows. The choice probability matrix is derived from the Gram Matrix, with elements

$$x_{ij} = \frac{\langle M_i, M_j \rangle \, [M_{ij} = 0]}{\kappa} \tag{9.31}$$

A $\langle M_i, M_j \rangle$ dot product gives the number of common ones in the two row vectors M_i and M_j. The normalisation constant κ may be the square of the maximal row sum, because every dot product is non-negative and it must not be bigger than this sum. A better choice may be the maximal $\langle M_i, M_j \rangle$, however. In the case of Binary Merge Squares the choice probability matrix can be defined by a matrix notation, based on Eq. 9.32. The only difference between the latter an Eq. 9.31 is that the $[M_{ij} = 0]$ restriction is expressed by a Kronecker product. The non-edge positions are 'masked out' by the entrywise product with the adjacency matrix of the complementer quotient graph $\bar{A}$. Using an entrywise product of matrices, this $\bar{A}$ is a suitable choice because it retains only those positions where $M_{ij} = A_{ij} \neq 0$, while the others will have a zero value.

$$X = \left(A \left(\frac{1}{\kappa} I \right) A^T \right) \circ \bar{A} \tag{9.32}$$

The choice strategy in the CC Merge Framework is defined by

$$\arg\max_{i,j} x_{ij} \tag{9.33}$$

The row choice probability function for the UC and CU Merge Frameworks. The UC and CU Merge Frameworks can apply this strategy as well, but the necessary coloured and uncoloured row sets must be kept and the choice probability matrix must contain additional zero elements to prevent merges between the rows having the same states, coloured or uncoloured, as seen in Figure 9.2. In the UC Merge Framework an uncoloured row is chosen by an arbitrary strategy, and based on the maximum dot product values a suitable coloured row is selected for the chosen uncoloured row.

Connection with the entrywise matrix norms. With a Binary Merge Square or Table this strategy can be expressed in terms of entrywise norms. A binary merge is the piecewise OR operation of the rows. Hence the number of ones decreases by the value of the dot product of two mergeable rows (see Eq. 9.34). In the case of Binary Merge Squares this decrease is twice this amount because the columns are also merged. The dot product maximisation strategy introduces a minimisation in the entrywise $1-$norm. To see this, let M be a Binary Merge Table and $M_{/rs}$ be the resulting Merge Table when the maximum dot product strategy is applied, where r and s are the row indices of the rows M_r and M_s selected by this strategy (Eq. 9.33). Eq. 9.35 helps explain why the norm decreases.

$$\left|M_{/rs}\right| = \textstyle\sum_{i \neq r, i \neq s} \sum_j M_{ij} + \left(\sum_j M_{rj} + \sum_j M_{sj} - \langle M_r, M_s \rangle \right) \tag{9.34}$$

$$\left(\textstyle\sum_{i \neq r, i \neq s} \sum_j M_{ij} + \sum_j M_{rj} + \sum_j M_{sj} \right) - \langle M_r, M_s \rangle = |M| - \langle M_r, M_s \rangle$$

The dot product maximisation introduces a minimisation in Eq. 9.34. This minimises the entrywise $1-$norm in the resulting Merge Matrix, as shown in Eq. 9.35. Hence the strategy can be turned into a norm minimisation where two vertices are chosen for a merge which minimises the norm, similar to that in Section 9.5.

$$\arg\left(|M| - \max_{r,s}\langle M_r, M_s\rangle\right) = \arg\min_{r,s}\left(|M| - \langle M_r, M_s\rangle\right) = \arg\min_{r,s}\left|M_{/rs}\right| \quad (9.35)$$

Since the resulting Merge Matrix is also a $\{0,1\}-$matrix an entrywise $1-$norm minimisation means a minimisation in every entrywise norm. Thus the entrywise $2-$norm (the Frobenius norm) is also minimised. The Frobenius norm may be calculated by the formula

$$\|M\| = \sqrt{\sum_{i=1}^{l_1}\sum_{j=1}^{l_2} M_{ij}^2} = \sqrt{\mathrm{tr}(M^T M)} = \sqrt{\sum_{i=1}^{\min\{l_1,l_2\}} \sigma_i^2} \quad (9.36)$$

where, l_1 and l_2 stand for the dimension of M. This lets us see the strategy from another aspect. There are different forms of the Frobenius norm (Eq. 9.36). However, they encode the same value, but their analysis better explains the principle behind the strategy. The first form gives the sum of the ones in the matrix; that is, the entrywise $1-$norm. The second is the sums of the row sums. This contains the maximum row sum employed separately in the Welsh-Powell strategy, but here it is only a component of the summation. The last expression represents the summation of the square of the σ_i singular values, where the principal singular value is just a component. Recall that the spectral norm minimisation strategy considers the principal singular value only. Both the spectral norm and the dot product strategies try to exploit a norm minimisation of the Binary Merge Matrices. The reason is that a final Merge Matrix, which corresponds to an optimal solution, has the minimal norm among the possible merge matrices which may come from the adjacency matrix, the initial Merge Matrix.

Remark. For a Binary Merge Square the third coefficient $-c_2$ of the characteristic polynomial (Eq. 3.13) of a quotient graph also gives the number of edges [9]; that is, the number of ones in the corresponding Merge Table (see Section 3.5). Reducing the number of non-zero elements can be a good heuristic to prevent the growth of non-zero elements which forbid possible merges. However, one can consider the zero elements of the Merge Matrix as well, since they supports the possible merges. Consequently we should deal with the ratio of the number of zero and non-zero elements which can characterise better our goal.

9.8 Cosine Strategy

Motivation. The cosine strategy was introduced by *the author* in [97], who demonstrated the efficiency of the strategy in several experiments [98; 100; 101]. Following

the dot product strategy, it is mainly intended for Binary Merge Models. The dot product strategy focuses on the evolution of the number of non-zero elements during successive merges and attempts to keep them as low as possible. To achieve this goal, the dot product strategy selects those two vertices for a merge which have the maximum dot product value. The goal of every Merge Algorithm is to make as many merges as possible because the number of merges is proportional to the quality of the solution as outlined in Section 9.7. Though the non-zero elements in a Merge Matrix frustrate the merges, the number of zeros assist them. Hence the cosine strategy takes the number of non-zero elements into account, but also considers the number of zeros present. It employs the maximum dot product strategy to determine the number of non-zero elements in the resulting Merge Matrix after a merge. In order to include the zero elements as well, the cosine strategy concentrates on the ratio of the zero and non-zero elements in the resulting Merge Matrix. Therefore in the row-pair choice this strategy combines the dot product of the two rows with the number of zero elements in the rows. Two rows are favoured in the selection if they have large dot product values and a large number of zero elements.

The row-pair choice probability function for the CC Merge Framework. A row has a large number of zero elements if it has few non-zeros. That is, the number of non-zero elements and the number of zero elements are inversely proportional. Therefore, to measure the number of zero elements in a row our strategy takes the reciprocal of the sum of a row. The sum of a row is provided by the vector entrywise $1-$norm. The reciprocal values are multiplied so as to have a common measure for the number of zeros of the two rows.

$$\frac{1}{|M_i|}\frac{1}{|M_j|} \tag{9.37}$$

Eq. 9.37 shows one of the components of our strategy, while Eq. 9.31 shows the other component, the dot product. The product of these components form the cosine strategy, which takes the non-zeros and zeros into account as well in the row-pair selection. In order to get suitable row-pairs, the product must be maximised. Hence, the row-pair selection of the cosine strategy is

$$\arg\max_{i,j} \frac{\langle M_i, M_j \rangle}{|M_i|\,|M_j|} \tag{9.38}$$

Since Binary Merge Matrices are $\{0,1\}$-matrices, the sum of the row elements is equal to the sum of the squares of the elements. Moreover, the square root of the sums does not change the selection in Eq. 9.38, hence

$$\arg\max_{i,j} \frac{\langle M_i, M_j \rangle}{\|M_i\|\|M_j\|} = \arg\max_{i,j} \frac{\langle M_i, M_j \rangle}{|M_i|\,|M_j|} \tag{9.39}$$

Where $|M_i|$ provides the sum of the row. Note that the square root of the sum of square of the elements gives the length of the vector. Based on this observation the row-pair choice probability function will be defined by the maximum cosine of two mergeable

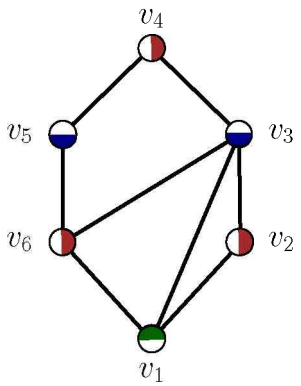
rows, like the following

$$x_{ij} = \frac{\langle M_i, M_j \rangle}{\|M_i\| \|M_j\|} [M_{ij} = 0] \tag{9.40}$$

Due to the cosine definition, the x_{ij} values always lie in the interval $[0, 1]$.

The row choice probability function for the UC and CU Merge Framework. The cosine strategy for the UC and CU Merge Frameworks may be defined in a similar way to the dot product strategy. In both cases this strategy is appropriate for the second choice. The condition is that the first row selection must be performed by another strategy, e.g. a greedy one. With the UC Merge Framework, the first row is selected from the uncoloured sub-merge-matrix, then the other row is selected from the coloured sub-merge-matrix using the maximum cosine strategy.

Remark. If a vertex is dominated by another vertex, i.e. when the neighbour set of one of them is a subset of the neighbour set of the other, then they can be coloured with the same colour in each optimal colouring. In our terminology they can be merged together before starting a colouring algorithm. The most obvious case is when they have the same common neighbours. Then their cosine value is one. If their neighbour sets are slightly different, then their cosine value will be high. Therefore our cosine formula appears to be meaningful in a merge process. A final Binary Merge Square is the adjacency matrix of a complete graph. Each row has $k - 1$ ones, and the dot product of each pair is $(k-2)$. Thus the cosine is value is $\frac{k-1}{(k-2)^2}$. This value is maximal if k is minimal. Hence the cosine strategy forces us to make k as low as possible; that is, it forces us to use as few colours as possible, which is the goal of minimal colouring. In Section 3.1.3 the zero blocks (i.e. independent sets) in the adjacency matrix are discussed, which form a solution (see Figure 9.8). Notice that the rows, which belongs to a zero block, are almost parallel. The next strategy is based on an optimisation which changes the zero entries in such a way as to get an almost parallel state of the appropriate rows.



	v_2	v_6	v_4	v_1	v_5	v_3
v_2	0	0	0	1	·	1
v_6	0	0	0	1	1	1
v_4	0	0	0	·	1	1
v_1	1	1	·	0	·	1
v_5	·	1	1	·	0	0
v_3	1	1	1	1	0	0

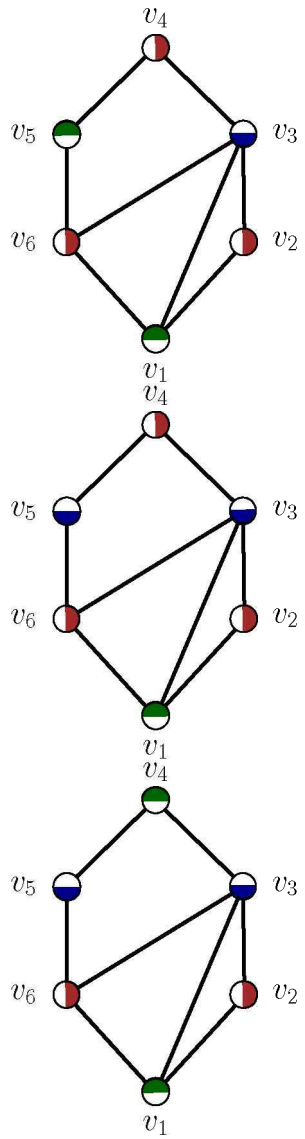
Figure 9.8: Zero blocks of the independent sets.

9.9 Zykov-tree and Lovász-theta strategy (enhanced cosine)

Motivation. This strategy was introduced for Binary Merge Squares only (for quotient graphs) by *the author* in [94; 102] based on the results of the authors of [51; 103; 110; 124; 161; 162]. Merge/colouring algorithms generate choice probability matrices for each step of the algorithm run. The matrix values represent probability values based on the algorithm strategy for how probable the merge of two rows is. A colouring matrix corresponds to a particular colouring. It describes whether two vertices are coloured with the same or different colours. Figure 9.9 shows all the optimal colouring matrices of a graph of Figure 2.1. All of them provide an exact merge probability matrix, where vertices in the same colour class have a probability value of one and differently coloured vertices have a value of zero. Unfortunately, none of them is available, because they form the solution of the minimal colouring problem. Although they are unavailable, their average can be approximated by a semi-definite program of Karger et al. [103]. The optimum of the semi-definite program is the so-called vector colouring number, the Lovász-theta [110] (see Section 3.4), but the optimum point is a matrix[10] $\hat{X}$, which approximates the average of the colouring matrices of optimal solutions. It is reasonable to call this matrix $\hat{X}$ because this will the basis of the choice probability matrix of the strategy, but the diagonal elements must be set to zero, otherwise they bias the selection.

The average of the optimal colouring matrices. The sum of the optimal colouring matrices contains very important information about the colouring (see Figure 9.9). An element of the sum matrix refers to the number of optimal colourings where two vertices got the same colour. Normalising the values by the number of optimal matrices results can result in a choice probability matrix. The normalisation turns the sum matrix into the average matrix of the optimal colouring matrices. In fact, the values of a choice probability matrix express the likelihood of the same colouring of two vertices. This can be characterised through the sum matrix. The values of the sum matrix are proportional to the relevant choice probabilities. Now let us consider the sum matrix of Figure 9.10. The zeros are still represented by dots, except for the $\{v_2, v_5\}$ position, there being a 0 instead of a dot. The dot positions are edges in the original graph, i.e. there are 1-s in the adjacency matrix. Although v_2 and v_5 are not connected, they are never coloured with the same colour in optimal colourings. Indeed, it is not hard to verify that choosing the same colour for them always leads to a non-optimal solution, a 4-colouring. But v_2 and v_6 are highly likely to get the same colour, because they share the same colour in all optimal solutions, as described by $X_{2,6} = 3$. Exploiting this observation, *the author* designed a Zykov-tree approach [161; 162]. Recall Section 4.3.1, where we introduced the Zykov-tree. Here there are two Zykov-steps, namely connection or contraction of two vertices. That is, a 1 addition to an appropriate Merge Square or performing a merge of two rows of the Merge Square. This strategy

[10] Rows of the matrix form the so-called vector colouring of the corresponding vertices of the graph.

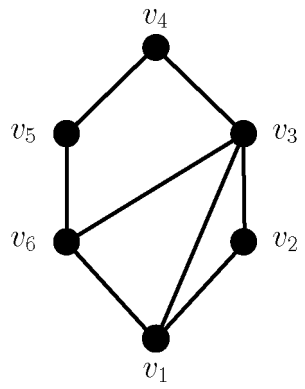


	v_2	v_6	v_4	v_1	v_5	v_3
v_2	1	1	1	·	0	·
v_6	1	1	1	·	·	·
v_4	1	1	1	·	·	·
v_1	·	·	·	1	1	·
v_5	0	·	·	1	1	·
v_3	·	·	·	·	·	1

	v_2	v_6	v_4	v_1	v_5	v_3
v_2	1	1	1	·	0	·
v_6	1	1	1	·	·	·
v_4	1	1	1	·	·	·
v_1	·	·	·	1	·	·
v_5	0	·	·	·	1	1
v_3	·	·	·	·	1	1

	v_2	v_6	v_4	v_1	v_5	v_3
v_2	1	1	·	·	0	·
v_6	1	1	·	·	·	·
v_4	·	·	1	1	·	·
v_1	·	·	1	1	·	·
v_5	0	·	·	·	1	1
v_3	·	·	·	·	1	1

Figure 9.9: The optimal colouring matrices of colourings. Here the rows and columns have been reordered for the sake of better clarity.



	v_2	v_6	v_4	v_1	v_5	v_3
v_2	3	3	2	·	0	·
v_6	3	3	2	·	·	·
v_4	2	2	3	1	·	·
v_1	·	·	1	3	1	·
v_5	0	·	·	1	3	2
v_3	·	·	·	·	2	3

Figure 9.10: The sum of the optimal colouring matrices of Figure 9.9

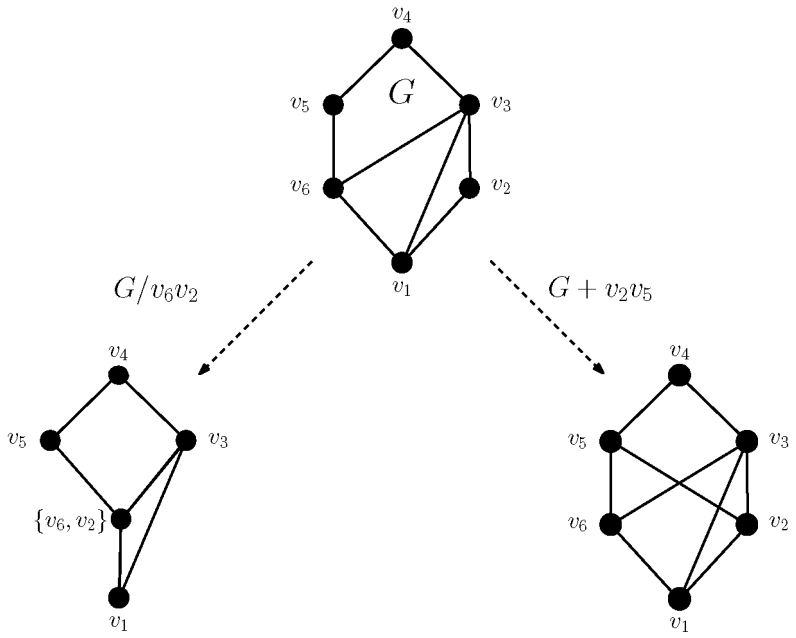


Figure 9.11: The Zykov-tree and Lovász-theta approach

combines the solution of a semi-definite optimisation of the Lovász-theta [103] with the Zykov-tree approach [162]. The optimisation produces an approximation matrix $\hat{X}$ for the average of the optimal colouring matrices. This $\hat{X}$ matrix may be a suitable basis for making a row-choice probability matrix, where two rows of a Binary Merge Square are merged if their row-pair choice probability in $\hat{X}$ is the largest. Furthermore, they are connected by an edge if their row-pair choice probability in $\hat{X}$ is the smallest and they are mergeable. An example of this can be found in Figure 9.11, where two Zykov steps are performed by the sum matrix of Figure 9.10. To define the strategy more precisely, we need to examine the approximation method of Karger et al. [103].

An approximation of the average of the optimal colouring matrices. Take an optimal colouring matrix example X_{opt} from Figure 9.9. This example is also shown in Figure 9.12(a). A colouring matrix $\tilde{X}$ is symmetric ($\tilde{X} = \tilde{X}^T$) and positive semi-definite ($\tilde{X} \succeq 0$), as shown in Section 3.1.4. In addition, $\tilde{x}_e = 0 \ \forall e \in E$; that is, a colouring matrix must have $0-$s in the edge positions where the adjacency matrix has $1-$s. Now decompose the example colouring matrix as follows: $X_{opt} = LL^T$, e.g. applying an Incomplete Cholesky Decomposition [69]. This is possible thanks to the symmetric and semi-definite properties of the colouring matrices. If X_{opt} is an $n \times n$ matrix, then the rows of L describe n sets of unit length vectors. However, some of these vectors may be the same (Figure 9.12(b)). Here, X_{opt} contains the dot products of the unit length vectors, i.e. the cosines of the angles of these vectors. Notice that the vectors in the decompositions define the colour assignments (c_i, v_i), where c_i is the $i-$th colour and v_i is the $i-$th vector. Finding an optimal colouring matrix is equivalent to finding an optimal colour assignment that minimises the number of colours k used in a proper colouring. One approach is to search the space of colouring matrices, where the number of $1-$blocks is to be minimised among all possible arrangements. Karger et

	v_2	v_6	v_4	v_1	v_5	v_3
v_2	1	1	1	·	0	·
v_6	1	1	1	·	·	·
v_4	1	1	1	·	·	·
v_1	·	·	·	1	1	·
v_5	0	·	·	1	1	·
v_3	·	·	·	·	·	1

(a) X_{opt}

	v_1	v_2	v_3	v_4	v_5	v_6
c_1	1	·	·	·	·	·
c_2	1	·	·	·	·	·
c_3	1	·	·	·	·	·
c_4	·	1	·	·	·	·
c_5	·	1	·	·	·	·
c_6	·	·	1	·	·	·

(b) L

Figure 9.12: An optimal colouring matrix example X_{opt} and its decomposition L into unit length vectors, where $X_{opt} = LL^T$.

al. turned the problem into an integer optimisation problem (see Eq. 9.41), where the number of colours k can be handled explicitly by retaining the positive semi-definiteness property of the transformed matrix [103]. That is

$$\chi = \min_k \{k : k\tilde{X} - J \succeq 0, \tilde{X} : \text{colouring matrix}\} \tag{9.41}$$

Here J is the matrix with all one elements and the minimisation is performed among all $\tilde{X}$ colouring matrices, and k is integer-valued. For the example colouring matrix X_{opt} (Figure 9.12(a)), the reformulated problem is shown in Figure 9.13 along with the normalisation factor $\frac{1}{k-1}$.

	v_2	v_6	v_4	v_1	v_5	v_3
v_2	$k-1$	$k-1$	$k-1$	-1	$\mathbf{-1}$	-1
v_6	$k-1$	$k-1$	$k-1$	-1	-1	-1
v_4	$k-1$	$k-1$	$k-1$	-1	-1	-1
v_1	-1	-1	-1	$k-1$	$k-1$	-1
v_5	$\mathbf{-1}$	-1	-1	$k-1$	$k-1$	-1
v_3	-1	-1	-1	-1	-1	$k-1$

(a) $kX_{opt} - 1$

	v_2	v_6	v_4	v_1	v_5	v_3
v_2	1	1	1	$\frac{-1}{k-1}$	$\frac{\mathbf{-1}}{\mathbf{k-1}}$	$\frac{-1}{k-1}$
v_6	1	1	1	$\frac{-1}{k-1}$	$\frac{-1}{k-1}$	$\frac{-1}{k-1}$
v_4	1	1	1	$\frac{-1}{k-1}$	$\frac{-1}{k-1}$	$\frac{-1}{k-1}$
v_1	$\frac{-1}{k-1}$	$\frac{-1}{k-1}$	$\frac{-1}{k-1}$	1	1	$\frac{-1}{k-1}$
v_5	$\frac{\mathbf{-1}}{\mathbf{k-1}}$	$\frac{-1}{k-1}$	$\frac{-1}{k-1}$	1	1	$\frac{-1}{k-1}$
v_3	$\frac{-1}{k-1}$	$\frac{-1}{k-1}$	$\frac{-1}{k-1}$	$\frac{-1}{k-1}$	$\frac{-1}{k-1}$	1

(b) $\frac{kX_{opt}-1}{k-1}$

Figure 9.13: Reformulations of the optimal colouring matrix X_{opt}.

A decomposition of Figure 9.13(b) describes 3 sets of unit length vectors. The cosine values of their pairwise angles are $\frac{-1}{k-1}$. The smaller the k, the larger the angle. The minimum is $k = \chi$. That is, for a $3-$chromatic graph (where $\chi = 3$) the angle is $120°$. This suggests an angle maximisation problem here. Eq. 9.41 just leads to an equivalent optimisation problem of the original minimal colouring problem. It attempts to find the colour assignment in an integer $\{0, 1\}$ vector space in accordance with Figure 9.12(b). Hence finding a solution requires as much effort as finding a solution when the problem is stated in the original form. Karger et al. following the observation of an angle maximisation, formulated a relaxed version of Eq. 9.41. This results in an efficient way to approximate the average of the optimal colouring matrices. In the relaxed problem

the $\{0, 1\}$ integer-valued feature of the $\tilde{X}$ is not required. However with the constraints, the edges must be contained in the relaxed model as well, and a decomposition of the matrix no longer describes unit vectors in an $n-$dimensional space. Nevertheless, it is reasonable to retain their unit length. Karger et al. also provided a relaxed problem in [103]. Their semi-definite optimisation program formulation is encapsulated by

$$\bar{\theta} = \min_t \{t : t\tilde{X} - J \succeq 0, \tilde{x}_{ii} = 1, \tilde{x}_e = 0 \; \forall e \in E\} \tag{9.42}$$

where $\tilde{x}_{ii} = 0$ guarantees the unit length of the vectors in the decomposition, $\tilde{x}_e = 0$ makes the appropriate edge constraints conform to the adjacency matrix $1-$s and t is a real-valued number. Solving the optimisation problem of Eq. 9.42 here makes use of the Lovász-theta number $\bar{\theta}$ introduced by Lovász in [110]. Section 3.4 describes an important property of the $\bar{\theta}$ number, namely $\omega \leq \bar{\theta} \leq \chi$. That is, the value is always a lower bound for the chromatic number, but an upper bound for the clique number. The optimum point, a semi-definite matrix, is the average of the optimal colouring matrices. A semi-definite optimisation solver can arbitrarily approach the optimum of Eq. 9.42, [103] providing an approximation for the average of the optimal colouring matrices. The standard semi-definite problem for Eq. 9.42 can be written down by introducing the matrix $Z = t\tilde{X} - J$:

$$\bar{\theta} = \min_t \{t : Z \succeq 0, z_{ii} = t - 1, z_e = -1 \; \forall e \in E\} \tag{9.43}$$

Let us denote the result of the optimum point of this Eq. 9.43 by Z_{opt}. Notice that Z_{opt} matrix must contain -1-s in the edge positions, where the adjacency matrix has 1-s and the optimal colouring matrices have 0-s.

The row-pair probability function for CC Merge Frameworks. Take $Z_{opt}+1$ to get zeros in the edge positions and set the main diagonal to zero, to get an appropriate basis for the choice probability matrix, i.e. $\hat{Z} = (Z_{opt} + 1) \circ (1 - I)$, where $(1 - I)$ entries are all ones except along the main diagonal, where it has zeros. *The author* in [94; 102] applied the values of the normalised $\hat{Z}$ matrix as a row-pair probability choice matrix in the CC Merge Framework. The normalisation which conforms to Eq. 9.2 is

$$X = \frac{\hat{Z} - \min \hat{Z}}{\max \hat{Z} - \min \hat{Z}} \tag{9.44}$$

The values of the $\hat{Z}$ matrix measure the colour similarity of the vertices and the matrix must contain 0-s in the edge positions; that is, the 0-s express dissimilarities. In addition, there can be negative values, therefore it is reasonable to apply a Zykov connection step for the appropriate vertices which correspond to the smallest negative value or the negative values leading to two sub-types of the strategy. Hence, *the author* added edges to each successive Binary Merge Square that is generated by a merge step in the following way. Take those (i, j) row-pairs for which $\hat{Z}_{ij} < 0$ is minimal (or $\hat{Z}_{ij} < 0$) and connect them by an edge. That is, place a 1 in the relevant position of the

adjacency matrix. Moreover, merge those two rows for which X_{ij} and hence $\hat{Z}_{ij}$ is maximal. These operations are repeated for each intermediate Merge Square; that is, the adjacency matrix for each successive quotient graph. However, since Eq. 9.44 is a suitable choice probability matrix, $\hat{Z}$ contains some more information to speed up the running time. $\hat{Z}_{ii}$ approximates $\bar{\theta}$ at the end of the optimisation. Learning something useful from the sum matrix structure of Figure 9.10, it is reasonable to merge not only the largest, but other rows as well which have large choice probability values. It means that one optimisation of Eq. 9.43 can result in more than one merge. That is, during a merge sequence, fewer semi-definite optimisations are required. Based on preliminary results, *the author* applied $\hat{Z}_{ij} > 0.5\bar{\theta}$ in [94; 102].

Parallel rows. In an optimal colouring matrix, those vertices which have parallel rows in the colouring matrix must be get the same colour. A row of a colouring matrix describes an exact colour similarity of the relevant vertex with the other vertices. A row of the average colouring matrix, and hence the choice probability matrix of Eq. 9.44, describes only an approximated colour similarity with the other vertices. If two rows of the choice probability matrix are parallel, then the two relevant vertices have the same relation to the others and they should be in the same colour class. Therefore cosine maximisation is a reasonable strategy for merging two rows. Moreover, the $-Z_{opt}$ matrix has 1-s in the same positions as the adjacency matrix, hence this observation is in agreement with the Cosine strategy of Section 9.8.

Improvement. Karger et al. [103] showed that in general, the average of the optimal colouring matrices is not a suitable way to get an optimal colouring. However, they were able to design a clustering algorithm which produces a semi-colouring where at most the quarter of the edges are coloured improperly. Based on the semi-colouring algorithm design in Lemma 3.1.1, they obtained the best known worst case including the first non-trivial bound. They were able to colour $k-$colourable graphs with at most $\min\{\mathcal{O}(n^{1-3/(k+1)}), \mathcal{O}(\Delta^{1-2/k})\}$ colours, where $n = |V_G|$ and Δ is the largest degree. During the merges n decreases and, in accordance with Section 9.4.1, Δ may decrease as well. Therefore it is reasonable to apply the average of the optimal colouring matrices with the merge approach.

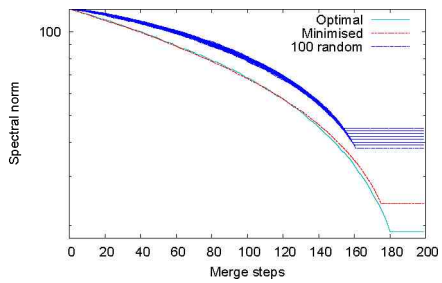
9.10 Merge Paths

Certain graph properties are evaluated during the selection of two rows for a Merge Operation, implicitly or explicitly. E.g. an explicit dot product maximisation strategy means an implicit norm minimisation. *The author* defined a general Merge Strategy using these properties (see *Juhos et al.* [101]). Here, an analysis of a supposed merge effect is performed. First, gather those graph properties into a vector which form the basis of the decision (e.g. spectral norm, i.e. the largest eigenvalue). One can take other eigenvalues as well to examine their evolution in the intermediate merge matrices during a merge sequence. Denote this vector by ξ. Determine which values are known in advance for the final merged graph. It is important to know these values because they will be the goal of this reformulated problem. Next, compute $\xi_{M^{[0]}}$ and $\xi_{M^{[n-k]}}$,

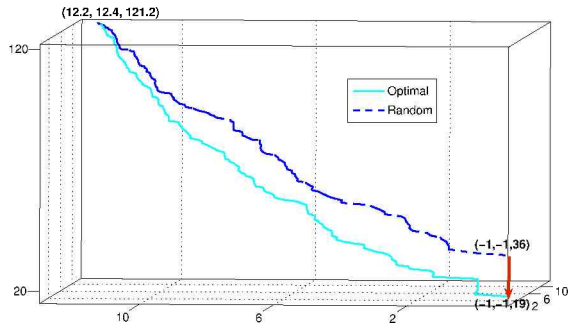
where $M^{[0]} = A_G$ and $M^{[n-k]} = A_{K_k}$, the adjacency matrix of G and K_k, respectively, in the case of Binary Merge Squares. Now the "only" task left is to find an appropriate merge sequence which corresponds to a path, a *Merge Path* from $\xi_{M^{[0]}}$ to $\xi_{M^{[n-k]}}$ in the vector space induced by $\xi_{M^{[t]}}$, where $M^{[t]}$ is an intermediate Merge Matrix in the t-th step. A Merge Path may be characterised by the sequence

$$\xi_{M^{[0]}},\ \xi_{M^{[1]}},\ \xi_{M^{[2]}}, \ldots, \xi_{M^{[n-k]}} \tag{9.45}$$

Figure 9.14(b) below shows an example of how the three largest eigenvalues ($\lambda_1 \geq \lambda_2 \geq \lambda_3$) of $M^{[t]}$ form different paths of 20- and 37-colourings in a three dimensional vector space. The start of the path is $(\lambda_1, \lambda_2, \lambda_3)_{M^{[0]}}$. The path ends at $(\lambda_1, \lambda_2, \lambda_3)_{M^{[n-k]}}$, which are known values in the case of Binary Merge Squares. The first value is trivial, because $M^{[0]} = A_G$ is given and the last is $(k-1, -1, -1)$, since the final merged graph G^{n-k} is a K_k complete graph on k vertices, where k is the number of colours used in the colouring[11]. Hence the goal is to get $(\chi - 1, -1, -1)$, which corresponds to a solution. An analysis of the paths helps us in the colouring process because we can identify and follow the optimal path. Now let us consider a simplified example in one dimensional space. Take the Binary Merge Square representation $A^{[t]}$ and let $\xi_{A^{[t]}} = \lambda_1(A^{[t]})$, i.e. the spectral norm of $A^{[t]}$. If we examine the initial Merge Square, we can see that $\lambda_1(A_G)$ is greater or equal than $\lambda_1(A^{n-\chi}) = \chi - 1$ (see [153]). Due to this fact the value of $\lambda_1(A^t)$ always decreases with each step, as shown in Figure 9.14(a).



(a) Spectral norm steepest descent minimisation. The ends of the curves have been extended (horizontal lines).



(b) An example 3D Merge Path of the three largest eigenvalues of a graph during a colouring.

Figure 9.14: Evolution of the eigenvalues along a merge sequence. The graph is a $20-$chromatic, equi-partite graph having 200 vertices with a 0.64 edge density from the peak of the phase transition. The spectral norm value of the final Binary Merge Square is $\chi - 1 = 19$ in the optimal case, otherwise it is bigger.

This path is responsible for determining the colouring and the end of the path $k-1$ defines the quality of the colouring. Unfortunately, the ideal path (between $\lambda_1(A^{[0]})$ and $\chi - 1$) is of course unknown; the task of colouring is to find this path. A colouring path takes $n-k$ colouring steps, that is $n-k$ merges. An optimal colouring needs $n-\chi$ steps,resulting in the longest step-series, while non-optimal colourings have shorter ones as they get stuck when no more merge are possible, i.e., $k > \chi$. If we consider the

[11] The minimum k is χ.

ideal path, which has the lowest bound and requires the most steps, then it should be below every possible non-optimal colouring path after a certain point. We can define a path in advance that has this property. A trivial path between the initial point and the end is a linear path, where $\lambda_1(A^{[t+1]})$ is derived from $\lambda_1(A^{[t]})$. The difference $\lambda_1(A^{[t+1]}) - \lambda_1(A^{[t]})$ should approximate $\frac{\lambda_1(A^{[0]})-\lambda_1(A^{[n-\chi]}))}{n-\chi}$. Non-linear paths can be defined by an analysis of more complicated graph properties and their behaviour. This approach can be applied to the row-pair choice probability values as well where a Merge Path may be defined by the maximal values of several choice probability functions. In this case it is not necessary to know the end of the path; the only requirement might be the component-wise increase of the path constituents. Since each merge brings a Merge Matrix closer to a candidate solution, the choice probability values must provide more confident choices.

9.11 Learning and clustering Merge Paths

The Merge Path approach allows the application of artificial intelligence methods in graph colouring, such as instance-based learning or clustering. Using a training set of graphs a learning algorithm (see *Juhos et al.* [95]) can learn certain Merge Paths that are associated with colouring steps (see Figure 9.14(b) or 9.14(a)). Actually, it is an approximation task, a curve fitting. First take a large set of generated graphs as training graphs with similar properties, e.g. $3-$chromatic equipartite graphs with a constant size (see Section 4.1). The approximation[12] of their optimal Merge Paths, i.e. that corresponds to an optimal merge sequence, may provide some useful information. Then this information can be used in the algorithm design. Take an unknown graph from the same category, e.g. a $3-$chromatic and equipartite graph with similar a size. The merges can be driven by the approximated learnt curve, where we generate that merge sequence which produces the closest path to the learnt Merge Path curve. Another possibility might be when an arbitrary merge sequence is performed and the distance is measured between the learnt Merge Path and the Merge Paths generated by the merge sequence. If a distance becomes critical, backtracking may be required. When the graph is derived from an unknown source, we do not know its category. A categorisation can be supported by clustering. Here, several training graph sets must be used as the basis for clustering. An arbitrary merge sequence of the unknown graph may have a characteristic shape. Hence, categorise this Merge Path derived from one or more arbitrary merge sequences using the training graph sets and a clustering algorithm. Based on the results of the clustering, the graph can be characterised e.g. by its chromatic number. Then like the above-mentioned learning task a colouring of the graph may be performed.

9.12 Evolutionary strategies

This section details uncoloured row-choice strategies for Merge Tables based on the evolutionary algorithm described in Section 4.2.7

[12] E.g. their average.

9.12.1 Finegrained fitness – the ζ fitness

An intuitive way of assessing the quality of a permutation of the vertices π as an uncoloured row choice strategy is by counting the number of rows remaining in the final Merge Table M. This is the same as the number of colours k_M used in the colouring of the graph which needs to be minimised. If we know that the optimal colouring is χ then we may normalise this fitness function such that $g(\pi) = k(\pi) - \chi$ or we can use a lower bound value of χ. This function gives a rather low diversity in the fitnesses of permutations because it cannot distinguish between two individuals that use the same number of colours. *The author* in [96] addressed this problem by introducing a new multiplier. This multiplier is based on the heuristic that we want to eliminate highly constrained rows in order to have a better chance of successful merges later on. This involves the merging of rows where many $1-$s are merged. Let $\zeta_{M(\pi)}$ denote the number of non-zeros in a final Merge Table, then the fitness function becomes $f(\pi) = (k_{M(\pi)} - \chi)\zeta_{M(\pi)}$, where $M(\pi)$ is the final Merge Table corresponding to the π permutation and a greedy merge/colouring scheme (see Section 10.1.1). This approach follows the entrywise norm optimisation of a Merge Table defined in Section 9.7.

9.12.2 Difficulty guided mutation

The evolutionary algorithm of Section 4.2.7 applies swap mutation as one of its variational operations. *The author* in [96] introduced a modified swap mutation. It always chooses the last merged row, which has few zero elements, and forces it to have an earlier position in the permutation in order to get a colour earlier. To accomplish this, it chooses at random a previous row identifier for a swap. The idea behind it is that these last merged rows are the most difficult to merge. The last rows are usually sparse ones, which have few non-zero elements. Though this strategy is simple, it actually proved quite useful in our experimental analysis in Section 10.2.2.

9.13 Summary

In this chapter we introduced several Merge Strategies which may be combined with a Merge Framework based on a Merge Model (see chapters 8 and 7). These strategies define the merging/colouring steps in an algorithm. Their motivation and analysis are provided, and a connection between them was also discussed. Due to the matrix-based Merge Models the strategies which apply them can be interpreted via different matrix properties such as matrix norms. Taking various matrix properties into account, we provided a new approach for the algorithm design, namely the Merge Path approach. We also showed that this approach allows one to apply machine learning and clustering methods for graph colouring.

Merge Strategies with different Merge Models and Merge Frameworks may result in different colouring algorithms. The next section describes possible combinations with experimental studies.

Chapter 10

Merge Algorithms

This chapter combines Merge Frameworks of Chapter 8 with the Merge Strategies of Chapter 9 to form a Merge Algorithm. The 'SUITABLE MODELS' section will describe which Merge Model supports the implementation of the algorithm in question. M will stand for a suitable Merge Model in the description, where M^{unc} is the uncoloured part of the Merge Matrix consisting the uncoloured rows, while M^{col} contains the coloured rows, as outlined in Section 7.2.

Existing algorithms can be expressed in a Merge Framework using one of the Merge Models. Benchmark algorithms of Section 4.2 as well as novel algorithms based on the strategies defined in Chapter 9 will be also described in this chapter. Description in a common way supports a structural analysis, and a fair performance comparison. Section 4.1 defines various benchmark graphs which form the basis of the comparison of the novel algorithms of the author *[94; 96–102] with the benchmark versions. Since the choices in the detailed algorithms are deterministic, just the unnormalised values of the choice probability functions will be considered.*

The following tokens will be used as abbreviations: BMT, IMT, BMS and IMS, where the B means 'binary', the I means 'integer'; the M is associated with 'merge', T and S stand for 'table' and 'square', respectively. Thus BMT means the 'Binary Merge Table' model. If a Merge Model should be emphasised in the notation of a framework, then the appropriate token appears on the top of the UC, CU or CC framework identifiers, e.g. $\overset{bmt}{UC}$. Similar to Section 4.2, the [.] operation makes a vector from the elements of a set, taking a natural order. Recall the sub and co-structures introduced in Section 7.2, which are the appropriate sums of the rows or columns of the relevant merge matrices denoted by μ. In the UC and CU Merge Frameworks the uncoloured and coloured parts of μ should be distinguished by the uncoloured and coloured sub-merge matrices: μ^{unc} and μ^{col}. Without ^{unc} or ^{col} indices, it belongs the whole Merge Matrix. There are four μ−s for each submatrix: left, right, top and bottom. Right μ_r and bottom μ_b co-structures count the non-zero elements of a row or column, respectively. Top μ_t and bottom μ_l are similar, but they contain the sum of each row and column, respectively. The sum of μ_t (or μ_l) co-structures is denoted by ζ_t and the sum of the bottom (or μ_r) co-structures are denoted by ζ_b. ζ_t is the sum of the entries of the relevant Merge Matrix, while ζ_b counts the non-zero elements of the Merge Matrix.

10.1 Benchmark algorithms in Merge Frameworks

This section describes the embedding of well-known benchmark algorithms of Section 4.2 into a suitable Merge Framework using an appropriate Merge Model by *the author*. A description of the algorithms in the common way supports their structural analysis.

10.1.1 Algorithms in the UC Merge Framework

The following benchmark methods can be described in the UC Merge Framework, where an uncoloured row is chosen followed by a coloured one for a merge. Hence it needs two choice functions: $choose-unc$ for the uncoloured row choice and $choose-col$ for the coloured row choice. In order to denote an algorithm in the UC Merge Framework let us introduce the following notation: $UC^{choose-col}_{choose-unc}$, where the $choose-unc$ denotes the uncoloured row choice strategy, while the $choose-col$ denotes the coloured one.

Greedy merge scheme

SUITABLE MODELS: BMT, IMT, IMS, BMS

The greedy colouring scheme of Section 4.2.2 fits nicely into the UC Merge Framework. Where the choice probability vector $\mathbf{x}$ is provided in advance by a strategy, then the first available colour c is assigned to the vertex chosen by the maximum value of $\mathbf{x}$. For tie breaking, when the choice is not exact, take the first vertex via a natural order of the vertices.

$UC^{(ext.\ strategy)}_{greedy}(A$ adjacency matrix $, \mathbf{x}\ \})$

1 $M \leftarrow A$
2 **repeat**
3 $u \leftarrow [\arg\max_i\{\ \mathbf{x_i}\ \}]_1$ // Choose by the maximum of choice prob. vector $\mathbf{x}$
4 $c \leftarrow \arg\min_i\{i : M^{col}_{ui} = 0\}$ // Choose a coloured row greedily
5 $M \leftarrow \mathrm{merge}(M, \{u, c\})$
6 $remove-component(\mathbf{x}, u)$ //Remove the $\mathbf{x}_u$ component
7 **until** M^{unc} is empty
8 **return** M

The greedy colouring scheme does not require any additional information during the colouring process. It performs the colouring using a predetermined order of the vertices by $\mathbf{x}$, which is provided by an external strategy. Therefore each Merge Model is suitable for making a colouring. One benefit of the application of the Merge Model is the decreased computational effort, which will be described later in Section 11.4.

Welsh-Powell

SUITABLE MODELS: BMT, IMT, IMS, (BMS)

We saw in Section 9.4.1 that the Integer Merge Models and the Binary Merge Table support this heuristic if it is defined in a dynamically varying merge environment. However, Welsh-Powell does not need to consider the varying conditions. The row-choice

probability function can be determined in advance. It is just defined by the degrees of the vertices. This can serve as an external strategy of the greedy merge scheme (see Section 10.1.1). Hence in this case any Merge Model can be applied. For demonstration purposes, we shall provide a Merge Algorithm which is defined like that in Section 9.4.1. This determines the relevant degrees during the colouring process.

$UC_{greedy}^{Welsh-Powell}(A$ adjacency matrix)
1 $M \leftarrow A$
2 **repeat**
3 $u \leftarrow [\arg\max_i\{ \mu_{li} \}]_1$ // Choose by maximum row sum
4 $c \leftarrow \arg\min_i\{i : M_{ui}^{col} = 0\}$ // Choose a coloured row greedily
5 $M \leftarrow \text{merge}(M, \{u, c\})$
6 **until** M^{unc} is empty
7 **return** M

Where μ_{li} is the $i-$th element of $\mu_l^{unc} = M^{unc}$ e, which gives the uncoloured row sum; that is, the degree of the relevant vertex in the original graph; while $\min_i\{i : M_{ui}^{col} = 0\}$ chooses the first available coloured row where the merge condition $M_{ui}^{col} = 0$ holds. $[.]_1$ decides the tie breaking cases if more than one maximal elements is found. It invariably chooses the first element in a natural order. This selection corresponds to the appropriate colour class represented by the coloured row. An uncoloured row merging into a coloured one means putting the uncoloured vertex into the appropriate colour class in the traditional sense.

Hajnal

SUITABLE MODELS: BMT, BMS, IMT, IMS
The Hajnal heuristic takes the vertices in reverse order by the principal eigenvector of the adjacency matrix and then performs a greedy colouring. As mentioned in Section 10.1.1, any Merge Model can be used as a basis for these algorithms, which does not take into account the varying environment during the colouring process. The Hajnal heuristic relies on a predefined choice probability vector, determined by the principal eigenvector $\hat{\mathbf{x}}$ (see Section 9.4.2) [1]; it does not change the strategy during the colouring. It requires a preliminary computation of the principal eigenvector as in the original definition in Section 4.2.4. The $[.]_1$ also decides the tie breaking cases here as it did (see Section 10.1.1), always choosing the first element. Similar to the greedy scheme in Section 10.1.1, it can be used with any of the Merge Models. It serves as an external strategy of the greedy merge scheme defined in Section 10.1.1 hence it is denoted by UC_{greedy}^{Hajnal}.

DSatur of Brèlaz

SUITABLE MODELS: IMT, IMS, (BMT, BMS)
DSatur uses the maximum saturation degree to choose an uncoloured vertex. The saturation degree is equal to the number of neighbour colours. For tie breaking it uses the degree of the vertices. After an uncoloured vertex is chosen, a greedy colour

[1] $M\hat{\mathbf{x}} = \lambda_{max}\hat{\mathbf{x}}$

assignment is applied (see Section 4.2.5). This heuristic takes into account varying state of the uncoloured vertices during the colouring process. Hence it can happily exploit the benefits of the Merge Models.

$\overset{ims}{UC}{}^{dsatur}_{greedy}(A$ adjacency matrix)

1 $M \leftarrow A$ // Let M be an Integer Merge Square
2 **repeat**
3 $\quad \mathbf{u} \leftarrow \arg\max_i\{\ (\mu_b^{col} \circ \mathbf{e}^{unc})_i\ \}$ // Choose by the max. uncol. top co-structure .
4 $\quad u \leftarrow [\arg\max_i\{\ (\mu_t \circ \mathbf{e}_\mathbf{u})_i\ \}]_1$ // Choose by the max. top co-structure.
5 $\quad c \leftarrow \arg\min_i\{i : M^{col}_{ui} = 0\}$ // Choose a coloured row greedily
6 $\quad M \leftarrow \text{merge}(M, \{u, c\})$
7 **until** M^{unc} is empty
8 **return** M

Here $\mathbf{u}$ contains the vector of the chosen uncoloured row indices according to the bottom co-structure of the coloured sub Merge Matrix μ_b^{col}, which defines the saturation degree of the vertices. It gives the number of neighbouring colours for each vertex since only the uncoloured rows/vertices are considered in the choice. The irrelevant part of the vector μ_b^{col} must be set to zero by $\mu_b^{col} \circ \mathbf{e}^{unc}$ because $\mathbf{u}$ may contain more than one component, i.e. references for uncoloured rows. DSatur applies a tie breaking by the vertex degrees. A column sum of the whole Integer Merge Table or Square M gives the relevant degree of an uncoloured vertex. Each column sum is placed in the top co-structure μ_t. Only the tie breaking positions of this vector are interesting; that is, the chosen uncoloured row indices $\mathbf{u}$. Therefore the irrelevant values of μ_t are set to zero by using the $\mathbf{e}_\mathbf{u}$ characteristic vector, where $\mathbf{e}_\mathbf{u}$ contains ones in the $\mathbf{u}$ positions, and zeros otherwise. Hence the entrywise product $\mu_t \circ \mathbf{e}_\mathbf{u}$ provides the necessary decision vector. Since the choice by this decision vector may still result in multiple uncoloured rows, the final tie breaking chooses the first element $[.]_1$. Keeping just the last tie breaking, the algorithm can use the Binary Merge Models as well. In the case of Binary Merge Tables or Squares, the bottom and top co-structures are the same ($\mu_b^{col} = \mu_t^{col}$) and can be calculated in the following way $\mu_b^{col} = (M^{col})^T\ \mathbf{e}$ which is the sum of the columns of the coloured sub Merge Matrix M^{col}. In order to restrict the calculation just for the uncoloured vertices, the co-structure must be multiplied[2] by $\mathbf{e}^{unc}$, which consists of ones only in the uncoloured vertex positions; that is, $\mu_b^{col} \circ \mathbf{e}^{unc}$. In the case of a Binary Merge Square it can be expressed by the equation

$$\mu_b^{col} \circ \mathbf{e}^{unc} = (M^{col})^T\ \mathbf{e}^{unc}$$

Furthermore, the top co-structure (the sum of the whole Integer Merge Matrix) gives the degree of the vertex in the original graph: $\mu_t = M^T\mathbf{e}$. It should be also restricted to uncoloured vertices so as to get a suitable choice probability vector[3] for the tie breaking cases; that is, $\mu_t \circ \mathbf{e}^{unc}$. For an Integer Merge Square it will be $\mu_t \circ \mathbf{e}^{unc} = M^T\mathbf{e}^{unc}$.

[2]Using elementwise product.
[3]Not normalised choice probability vector.

Evolutionary algorithm – standard fitness

SUITABLE MODELS: BMT, BMS, IMT, IMS

The evolutionary algorithms of Section 4.2.7 can serve as an external uncoloured row choice strategy for the greedy algorithm scheme defined in the UC Merge Framework in Section 10.1.1. Then k_M counts the number of rows of the final Merge Matrix M got in the greedy colouring process. This metaheuristics approach maintains a set of vertex permutations Π (population) via its swap mutation and order based crossover operators modifying the candidate solutions of Π or creating new ones. The selection operator is a $2-$tournament selection, which keeps the population in a steady-state; that is, the number of elements remains constant while the algorithm is running. In order to measure the goodness of a candidate solution (i.e. a permutation), it performs a simple measurement; it counts the number of colours used in the greedy colouring. The number of colours is equal to the number of rows that remain in the final Merge Matrix of the UC_{greedy} scheme described in Section 10.1.1. Furthermore, $\hat{\chi}$ is a normalisation constant, which is a lower bound of the chromatic number. In an experimental study $\hat{\chi}$ may be the χ, and hence the zero value of a fitness f can terminate the running algorithm, when an optimal solution is found. Otherwise, the stop condition depends on a certain time limit, which can be determined in various ways, e.g. by counting the number of fitness evaluations.

$UC^{EA}_{greedy}(A$ adjacency matrix $)$

1 $\Pi \leftarrow random\ permutations(population\ size)$
2 **while** $termination\ condition$
3 **do**
4 **for** $\pi \in \Pi$ // Evaluate each permutation
5 **do**
6 $M \leftarrow UC_{greedy}(A, \pi)$ // M is a final Merge Matrix
7 $f(\pi) \leftarrow k_M - \hat{\chi}$ // Fitness [a]
8 $\Pi = \Pi \cup swap(\Pi, p_{mut}) \cup ox2(\Pi, p_{xover})$
9 $\Pi = tour_2(\Pi, f)$
10 $\pi \leftarrow best(\Pi, f)$
11 **return** $UC_{greedy}(A, \pi)$

[a] $\hat{\chi}$ is a lower bound of χ.

Since this evolutionary algorithm uses the UC_{greedy} for colour assignment and it does not exploit any additional feature of the Merge Models, all Merge Models will be suitable for the implementation.

Evolutionary algorithm – Stepwise adaptation of weights (SAW)

SUITABLE MODELS: EXTENSIONS OF THE IMT OR IMS

Here it applies an improper colouring scheme, so Merge Models cannot describe this scheme. The models may be extended to handle improper colourings as well by allowing

the merges for the rows where the merge condition is not satisfied. The SAW algorithm requires the number of violated constraints at the end of a colouring. Integer Merge Models do not lose any edges.

10.1.2 Algorithms in the CU Merge Framework

This Merge Framework supports the so-called independent set approach, described in Section 4.2.1. Only the Erdős heuristic apply this approach among the benchmark algorithms. In the CU Merge Framework, a coloured row is chosen followed by an uncoloured one for a merge. It requires the same two choice functions as the UC Merge Framework: $choose - col$ for the coloured row choice and $choose - unc$ for the uncoloured row choice. However, here they are applied in reverse order. In order to define an algorithm in the CU Merge Framework let us introduce the following notation: $CU^{choose-col}_{choose-unc}$, where the $choose - col$ denotes the coloured row choice strategy, while the $choose - unc$ denotes the uncoloured one.

Erdős

SUITABLE MODELS: IMT, IMS (BMT, BMS)
The Erdős heuristic takes the first colour and assigns it to the vertex v that has the minimum degree. Vertex v and its neighbours are then removed from the graph. We apply the algorithm in the remaining sub-graph in the same fashion until the sub-graph becomes empty, then take the next colour and use the algorithm for the non-coloured vertices and so on until each vertex is assigned a colour.

$CU^{ims,\,greedy}_{Erdös}(A$ adjacency matrix)

1 $M \leftarrow A$
2 $u \leftarrow [\,]$ // Empty choice of an uncoloured row index
3 **repeat**
4 $\quad c \leftarrow \arg\min_i\{i : M^{col}_{ui} = 0\}$ // Choose the earliest available coloured row
5 $\quad u \leftarrow [\arg\min_i\{\ (\mu^{unc}_b \circ \mathrm{e}^{unc})_i : M^{col}_{ci} = 0\ \}]_1$ //Choose by min. uncol. degree
6 $\quad M \leftarrow \mathrm{merge}(M, \{u, c\})$
7 **until** M^{unc} is empty
8 **return** M

Where μ^{unc}_b contains the uncoloured degrees. Similar to the Section 10.1.1, μ^{unc}_b can be defined by $\mu^{unc}_b = (M^{unc})^T\mathrm{e}$. Moreover, if we choose just the values for the uncoloured vertices further processing is required:

$$\mu^{unc}_b \circ \mathrm{e}^{unc} = (M^{unc})^T\mathrm{e}^{unc}$$

In the case of Merge Tables e^{unc}_G must be used, the characteristic vector of the uncoloured vertices in the original graph, which contain ones only in the uncoloured positions and zeros elsewhere. Here $\min_i\{i : M^{col}_{ui} = 0\}$ always chooses the last coloured

row index. When $c = [\ ]$, (i.e. there can be no more mergeable uncoloured row with this coloured row), the merge is a simple marking of the chosen uncoloured row M_u in the coloured rows; that is, it places it into the coloured sub Merge Matrix.

10.2 Novel Merge Algorithms

This section describes algorithms which arise from a combination of a Merge Framework of Chapter 8 and a Merge Strategy of Chapter 9. They were introduced by *the author* in [94; 96–102]. Since the benchmark algorithms are defined in a Merge Framework in Section 10.1, their structural analysis and comparison with these novel methods can be performed in the same way. Structural analysis will be described in Chapter 11, while an experimental comparison will be provided in this section. The experimental comparisons are based on well-known benchmark graphs and generated random equipartite graphs on 200 vertices according to Section 4.1 in the phase transition region (see Section 3.8) where the problems become hard.

10.2.1 Algorithm in the UC Merge Framework – uncoloured row choice strategies

This section describes two novel algorithms introduced by *the author* in [96]. These are evolutionary algorithms based on the strategies defined in sections 9.12.1 and 9.12.2. The algorithms are combined with the greedy merge scheme of Section 10.1.1. They attempt to find a suitable permutation of the rows to achieve a minimal colouring by the greedy merge scheme. The permutations define the appropriate row choice vector that will be given to the greedy Merge Algorithm. An experimental comparison will be provided with two well-known benchmark algorithms which were described in Section 4.2.

Evolutionary algorithm – the ζ fitness

SUITABLE MERGE MODELS: BMT, IMT

This algorithm is based on a standard evolutionary algorithm (see Section 4.2.7) that introduces a new fitness calculation, namely the ζ fitness, based on a Merge Table Model as described in Section 9.12.1. In order to achieve this goal a Merge Algorithm is necessary to provide a final Merge Table; it is a simple greedy merge based on Section 10.1.1. The evolutionary algorithm applies a swap mutation and $2-$point order based crossover to change the permutations. A $2-$tournament selection is then applied to keep the number of permutations contant. Similar to the Section 10.1.1, the evolutionary algorithms serve as an external uncoloured row choice strategy in a UC Merge Framework.

$UC^{EA^{\zeta}}_{greedy}(A$ adjacency matrix)
1 $\Pi \leftarrow random\ permutations(population\ size)$
2 **while** $termination\ condition$
3 **do**
4 **for** $\pi \in \Pi$ // Evaluate each permutation
5 **do**
6 $M \leftarrow UC_{greedy}(A, \pi)$ // M is a final Merge Matrix
7 $f(\pi) \leftarrow (k_M - \hat{\chi})\zeta_M$ // Fitness [a]
8 $\Pi = \Pi \cup \mathbf{swap}(\Pi, p_{mut}) \cup ox2(\Pi, p_{xover})$
9 $\Pi = tour_2(\Pi, f)$
10 $\pi \leftarrow best(\Pi, f)$
11 **return** $UC_{greedy}(A, \pi)$

[a] $\hat{\chi}$ is a lower bound of χ.

Evolutionary algorithm – difficulty guided mutation

Suitable Merge Models: bmt, imt

Here, a modification of the evolutionary algorithm of Section 10.2.1 is provided by altering the swap mutation to the difficulty guided mutation of Section 9.12.2.

$UC^{EA^{\zeta,dgs}}_{greedy}(A$ adjacency matrix)
1 $\Pi \leftarrow random\ permutations(population\ size)$
2 **while** $termination\ condition$
3 **do**
4 **for** $\pi \in \Pi$ // Evaluate each permutation
5 **do**
6 $M \leftarrow UC_{greedy}(A, \pi)$ // M is a final Merge Matrix
7 $f(\pi) \leftarrow (k_M - \hat{\chi})\zeta_M$ // Fitness [a]
8 $\Pi = \Pi \cup \mathbf{dgs}(\mathbf{\Pi}, \mathbf{p_{mut}}, \mathbf{M}) \cup ox2(\Pi, p_{xover})$
9 $\Pi = tour_2(\Pi, f)$
10 $\pi \leftarrow best(\Pi, f)$
11 **return** $UC_{greedy}(A, \pi)$

[a] $\hat{\chi}$ is a lower bound of χ.

10.2.2 Experiments

This section details the experimental results obtained from running two Merge Algorithms which apply uncoloured row choice strategies introduced by *the author* in [96]. Row choice vectors are encoded in permutations of the vertices. These permutations are changed by applying following evolutionary algorithms.

Algorithms introduced by *the author* in [96]

$UC^{EA^{\zeta}}_{greedy}$ – ζ fitness: the algorithm is based on the Binary Merge Table Model that utilises the greedy merge scheme UC_{greedy}. For a fitness calculation of a π permutation it uses the ζ fitness $f(\pi) = (k_M - \chi)\zeta_M$, as described in Section 9.12.1. It applies a Binary Merge Table Model.

$UC^{EA^{\zeta,dgs}}_{greedy}$ – ζ fitness and difficulty guided mutation. This variant applies OX2 with a probability of 0.3 and then always applies a heuristic mutation operator that is similar to the simple swap mutation; but it always chooses a vertex related to the last merged row and forces it take earlier position in the permutation. To accomplish this, it chooses at random a previous row identifier for a swap. The idea is that these last merged rows are the most difficult to merge. It then applies a Binary Merge Table Model.

Benchmark algorithms

The following algorithms served as a reference in our experimental comparison.

$UC^{bt-dsatur}_{greedy}$: the DSatur heuristic embedded into the UC Merge Framework, as described in Section 10.1.1 using, a backtracking for exhaustive search of the permutation space (see Section 4.3 and 3.6). It utilises an Integer Merge Table Model.

EA_{saw}: an evolutionary algorithm that applies a stepwise adaptation of weights heuristic defined in Section 4.2.8. It does not use any Merge Model.

An evolutionary algorithm with standard fitness (see Section 4.2.7) was not included in the benchmark set of the algorithms, because experiments by *Juhos et al.* in [99] showed that an evolutionary algorithm with the ζ fitness clearly outperforms this one, which has a standard fitness. Furthermore, for a fair comparison we used that variant of DSatur which is embedded into the UC Merge Framework, otherwise its results are much worse as described in Section 11.4. EA_{saw} uses improper colouring so the current Merge Models are unsuitable for them, but their participation in the test is useful because the EA_{saw} method proved very efficient on random $3-$chromatic equipartite graphs in [52; 145].

Means of Comparisons

The performance of an algorithm can be characterised by its effectiveness and efficiency in solving a problem instance. The first is measured using the success ratio, which is the amount of runs where an algorithm has found the optimum divided by the total number of runs. The second is measured by keeping track of how many constraint checks are being performed on average for a successful run. This measure is independent of hardware and programming language as it counts the number of times an algorithm requests information about the problem instance, e.g. it checks whether an edge exists between

two vertices in the graph. This check, or rather the number of times it is performed, comprises the largest amount of time spent by these constraint solvers [52; 145]. A constraint check is defined, for an algorithm, as a check of whether the colouring of two vertices is allowed (satisfied) or not allowed (violated). The evolutionary algorithms are all stochastic algorithms. Therefore we performed 10 independent runs with different random seeds for each problem instance. The number of constraint checks were then averaged over these 10 runs. The exhaustive search method, $UC_{greedy}^{bt-dsatur}$ needs just one run.

Algorithm settings

The stop condition for an algorithm is that either an optimum has been found or that the $1\,500\,000$ limit of constraint checks has been reached. The latter means that the run was unsuccessful, i.e. an optimal colouring was not found. For evolutionary algorithms it means that a permutation π exists with $f(\pi) = 0$ fitness. Furthermore, for the evolutionary algorithms the population size is set to 100 for graphs having at least 150 vertices, otherwise it is set to 20. The evolutionary algorithms performs the $\mathrm{OX2}$ 1-point order based crossover with a $p_{xover} = 0.6$ probability and with a probability of $p_{mut} = 0.3$ for the simple swap mutation. These probability values were determined by preliminary tests on random graphs.

Benchmark graphs

There are two sets: the standard benchmark set of the DIMACS Challenge was introduced in Section 4.1 and the class of random $3-$chromatic equipartite graphs on 200 vertices $\mathcal{G}_{eq,n=200,0.02\leq p_e\leq 0.06,k=3}$ generated by using Culberson's generator [44] in the phase transition region (see Section 3.8). It consists of 9 groups of graphs with different edge probabilities p_e, where each group has 25 instances. The edge probability p_e is changed from 0.020 to 0.060 in steps of 0.005, resulting in 9 groups. Further details about these graphs and about the phase transition can be found in sections 3.8, 4.1 and 3.7.

Results

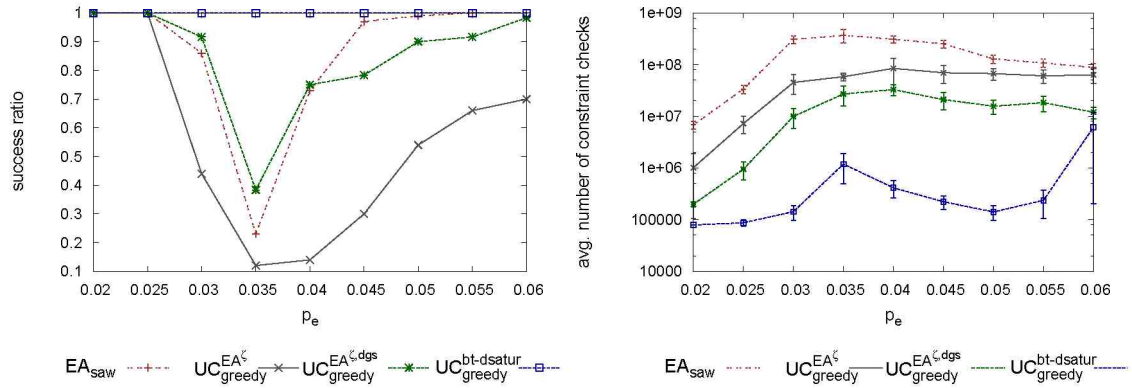
Analysing Table 10.1, for large graphs the novel algorithms, the $UC_{greedy}^{EA^{\zeta}}$ and the $UC_{greedy}^{EA^{\zeta,dgs}}$ are much faster than the benchmark algorithms. Note that the Merge Models reduce the number of constraint checks quite considerably (see Section 11.4) for the miles and queens graphs, where the difficulty guided mutation outperforms the simple swap mutation. Moreover, the latter is not always able to find a solution for two of the queen graphs as the success ratio of this algorithm is less than one. In Figure 10.1 we can clearly see that $UC_{greedy}^{bt-dsatur}$ is the best algorithm here as it always finds a solution and it uses almost the minimum number of constraint checks to achieve it. The results for the four algorithms in Figure 10.1(b) are significantly different and allow us to give a clear ranking on the efficiency for the three algorithms. All evolutionary algorithms

Table 10.1: Average number of constraint checks required for solving various problem instances. Entries with "–" refer to where the algorithm never found the chromatic number, while in every other case the success ratio is one. The last three entries are used to highlight the differences between the two mutation operators for the swap and the difficulty guided (dgs) mutations.

Graph	$\|V\|$	$\|E\|$	χ	$UC_{greedy}^{bt-dsatur}$	EA_{saw}	$UC_{greedy}^{EA^{\zeta}}$	$UC_{greedy}^{EA^{\zeta,dgs}}$
mulsol.i.1	197	3 925	49	811 595	6 265	5 964	8 525
mulsol.i.2	188	3 885	31	485 644	21 707	4 110	5 667
mulsol.i.3	184	3 916	31	461 953	51 042	4 874	5 619
mulsol.i.4	185	3 946	31	467 398	128 130	4 084	5 606
mulsol.i.5	186	3 973	31	472 872	11 120	4 141	5 536
zeroin.i.1	211	4 100	49	1 056 595	13 165	6 670	8 040
zeroin.i.2	211	3 541	30	641 583	65 053	11 870	4 942
zeroin.i.3	206	3 540	30	603 978	52 493	22 556	11 197
anna	138	493	11	105 811	15 579	2 903	1 242
david	87	406	11	40 772	56 872	9 957	2 493
huck	74	301	11	27 122	1 210	788	1 015
jean	80	254	10	29 101	11 390	746	949
miles500	128	1 170	20	147 922	9 724 950	191 011	20 398
miles750	128	2 113	31	204 871	7 922 930	946 683	103 376
miles1000	128	3 216	42	244 886	15 476 000	1 551 235	164 312
miles1500	128	5 198	73	329 361	886 155	167 487	67 721
myciel6	95	755	7	27 807	5 920	708	955
myciel7	191	2 360	8	134 956	52 997	4 074	3 245
games120	120	638	9	60 777	3 227	1 492	1 926
queen5_5	25	160	5	1 665	8 835	4 630	2 000
queen6_6	36	290	7	320 063	–	740 550	139 711
queen7_7	49	476	7	1 176 441	3 195 320	6 326 912[1]	744 273
queen8_8	64	728	9	150 000 150	–	30 412 779[2]	9 558 255

[1] A success ratio of 0.9

[2] A success ratio of 0.4



(a) Success ratio vs p_e. Note that $UC_{greedy}^{bt-dsatur}$ always finds a solution

(b) Average number of constraint checks with 95% confidence intervals

Figure 10.1: Results for 225 random equipartite graph $3-$chromatic problems of size 200, where for each problem instance 10 independent runs are performed.

show a sharp dip in the success ratio in the phase transition (see Figure 10.1(a)), which is accompanied by a rise in the average number of constraint checks. $UC_{greedy}^{EA^{\zeta,dgs},}$ starts out over 34 times faster than the EA_{saw} benchmark algorithm. When the number of edges increases this difference decreases to 7 times as fast. $UC_{greedy}^{bt-dsatur}$ seems to have the least problems with this problem set. It performs well on $3-$chromatic graphs, but its performance degrades if the chromatic number of the graph instances increase as shown in Section 10.2.9 later on.

10.2.3 Conclusions

We verified the efficiency of the two new colouring algorithms of *the author* [96] by performing an empirical comparison on two test suites. The results from the DIMACS test suite show a performance in speed and accuracy that is quite favourable, especially on large real-world problem instances with 400 vertices. For larger problem instances it is much faster than EA_{saw} and $UC_{greedy}^{bt-dsatur}$. However in the second test, where we looked at equipartite graphs during the phase transition, the success ratio shows the typical dip we often observe for stochastic algorithms, but the new algorithms yielded better results. For difficult equipartite graphs, i.e. those that lie near the peak of the phase transition, it is less effective than $UC_{greedy}^{bt-dsatur}$, but it is faster than EA_{saw}.

10.2.4 Experiments done in the UC Merge Framework – coloured row choice strategies

In Section 10.2.2 two novel efficient Merge Algorithms were given. These algorithms applied the greedy merge scheme to perform colouring. In this section we will present non-greedy colour assignments. In the UC Merge Framework it corresponds to changing of the greedy coloured row choice scheme to another one. Two novel coloured row choice strategies of *the author* [97] will be examined here and compared with the

greedy row choice scheme.

Dot Product and Cosine strategies in the UC Merge Framework

Sections 9.7 and 9.8 described two novel strategies called the Dot Product and Cosine strategies. These strategies support row-pair choices; that is, algorithms in the CC Merge Framework. However they can be work as a second row choice strategies, if one is selected by another row choice strategy. In the UC Merge Framework the first choice is the uncoloured row choice and the second is the coloured one. Hence, they will choose coloured rows. The uncoloured row choice will be tackled by an evolutionary algorithm with ζ fitness (see Section 10.2.1).

$UC^{ext.\ strat.}_{dotprod}(A$ adjacency matrix $, \mathbf{x}\ \})$

1 $M \leftarrow A$
2 **repeat**
3 $\quad u \leftarrow [\arg\max_i\{\ \mathbf{x_i}\ \}]_1$ // Choose by the maximum of $\mathbf{x}$
4 $\quad c \leftarrow \arg\max_i\{\langle M_i, M_u\rangle : M^{col}_{ui} = 0\}$ //Choose a col. row by **max. dot prod.**
5 $\quad M \leftarrow \text{merge}(M, \{u, c\})$
6 $\quad remove - component(\mathbf{x}, u)$ //Remove the $\mathbf{x}_u$ component
7 **until** M^{unc} is empty
8 **return** M

$UC^{ext.\ strat.}_{cos}(A$ adjacency matrix $, \mathbf{x}\ \})$

1 $M \leftarrow A$
2 **repeat**
3 $\quad u \leftarrow [\arg\max_i\{\ \mathbf{x_i}\ \}]_1$ // Choose by the maximum of $\mathbf{x}$
4 $\quad c \leftarrow \arg\max_i \left\{ \frac{\langle M_i, M_u\rangle}{\|M_i\|\|M_u\|} : M^{col}_{ui} = 0 \right\}$ //Choose a coloured row by **max. cos.**
5 $\quad M \leftarrow \text{merge}(M, \{u, c\})$
6 $\quad remove - component(\mathbf{x}, u)$ //Remove the $\mathbf{x}_u$ component
7 **until** M^{unc} is empty
8 **return** M

An external strategy provides $\mathbf{x}$ as a choice probability vector, which may be unnormalised too. The uncoloured row is chosen by this taking the position of its maximum value. Then either the Dot Product strategy or the Cosine strategy selects a coloured row to merge with this uncoloured row. In the coloured row choice calculation the $\|M_u\|$ in the denominator is a constant, hence it can be removed and the following can be applied instead: $c \leftarrow \arg\min_i \left\{ \frac{\langle M_i, M_u\rangle}{\|M_i\|} \right\}$.

10.2.5 Experiments

Firstly, the novel coloured row choice strategies are compared with the greedy coloured row choice strategy. Here the evolutionary algorithm of Section 10.2.1 selects the uncoloured rows. The experimental setup is the same as that outlined in Section 10.2.2.

Then an extended experiment will be provided, where other benchmark algorithms are also included in the comparison. In addition, it contains other graph types and other test run results of the algorithms. The evolutionary algorithms all correspond to that described in Section 10.2.1. They use the ζ fitness function based on a Binary Merge Table.

Algorithms introduced by *the author* in [97]

$UC^{EA^{\zeta}}_{dotprod}$: the evolutionary algorithm of Section 10.2.1, provides the external strategy for $UC^{ext.\ strat.}_{dotprod}$. It applies a Binary Merge Table Model.

$UC^{EA^{\zeta}}_{cos}$ the evolutionary algorithm of Section 10.2.1, provides the external strategy for UC_{cos}. It applies a Binary Merge Table Model.

Benchmark algorithms

$UC^{EA^{\zeta}}_{greedy}$: the evolutionary algorithm provides the external strategy for $UC^{ext.\ strat.}_{greedy}$ (see Section 10.2.1). It applies a Binary Merge Table Model.

The basis of the comparison made here, is the same $3-$chromatic benchmark graph set as that given in the Section 10.2.1. The algorithm settings and the mean of the comparison are also similar to those stated in Section 10.2.1.

Results

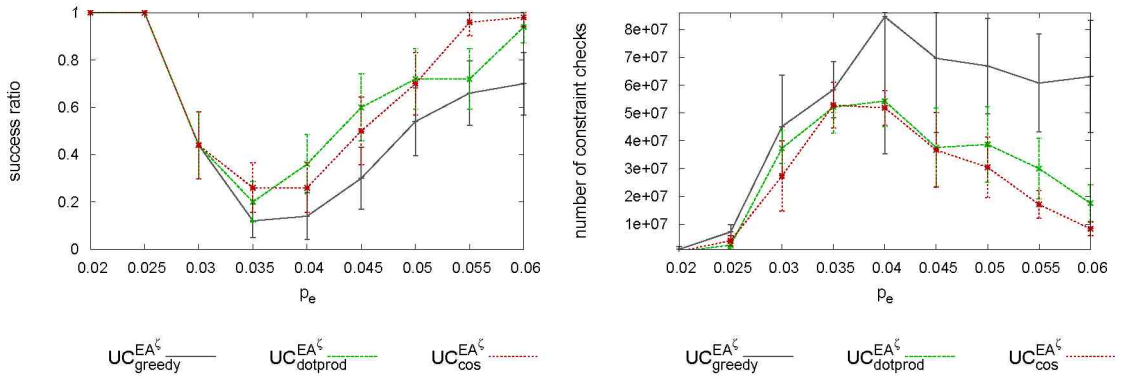
The two novel strategies that utilise details about the colouring of the graph made so far are shown in Figure 10.2 together with the simple greedy strategy. Here we notice a clear improvement in both the efficiency and effectiveness relative to the simple greedy strategy. In particular, the search effort needed for denser graphs is less. Furthermore, the confidence intervals for this range are small and non-overlapping. These two approaches furnish a much more robust algorithm for solving graph k-colouring problems.

10.2.6 Extended experiments

Juhos et al. in [100; 101] carried out other investigations of these strategies. They compared the methods with the backtracking version of the DSatur algorithm $UC^{bt-dsatur}_{greedy}$, in accordance with Section 10.2.2, using various random equipartite graphs. The algorithms settings were the same as those in the experiments described in Section 10.2.2.

Benchmark graphs

The test set consists of $k-$colourable equipartite graphs with 200 vertices, where k is set to $3, 5, 10$ and 20 ($\mathcal{G}_{eq,n=200,0.02\leq p_e\leq 0.98,k\in\{3,5,10,20\}}$) using Culberson's generator [44]. For $k = 20$, ten vertices will form a colour set, hence we will not use any larger



(a) Success ratio with 95% confidence intervals (b) Average number of constraint checks with 95% confidence intervals

Figure 10.2: Results for 225 random equipartite graph 3−colouring problems of size 200, where for each problem instance 10 independent runs were performed.

number. The edge probability of the graphs is varied in a region called the phase transition. Using this test set we can ensure a fair comparison of the algorithms, since this set contains problems ranging from the easy to the most difficult. Moreover, we would like to avoid any comparison on some chosen real-life problems where the selection method can determine the outcome of the comparison of the performance (see [41]). The set consists of groups where each group is a $k-$chromatic with 20 unique instances.

Means of Comparisons

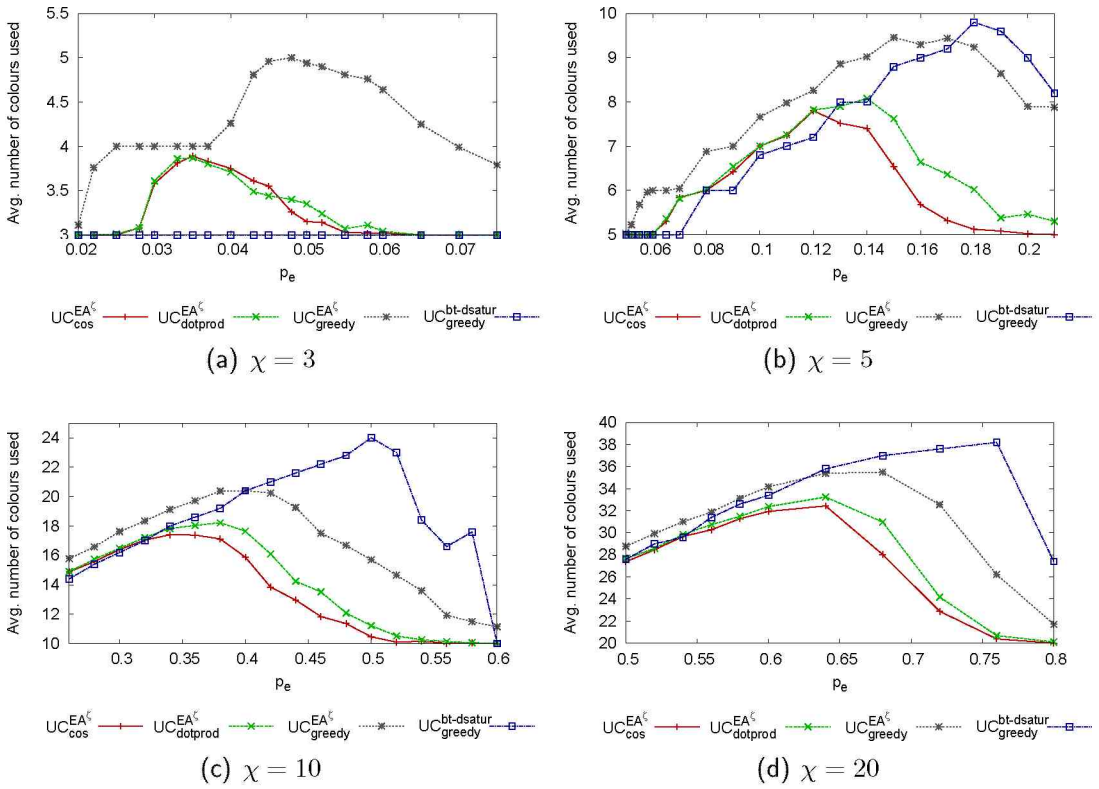
On each instance we performed ten independent runs and calculated averages of the number of colours used. These averages were further averaged over each graph instances which had the same edge probability, i.e. over the edge probability groups. Confidence intervals were also calculated, but they just confirmed our anticipated results, hence they were not plotted in the figures here for the sake of clarity.

Results

Figure 10.3 shows that the Cos heuristic performs well, especially for larger k, while the Dot Product is a close second. DSatur is the strongest algorithm on 3-colourable graphs, where it always finds the optimum number of colours. However, backtracking can help on very sparse graphs, DSatur quickly gets the last position as the chromatic number and hence the edge density grows.

10.2.7 Conclusions

By comparing the different strategies on several hard-to-solve problems, we showed how employing coloured-row choice strategies can improve the convergence speed of the evolutionary algorithm. Furthermore, the two novel strategies, i.e. the Dot Product



(a) $\chi = 3$ (b) $\chi = 5$

(c) $\chi = 10$ (d) $\chi = 20$

Figure 10.3: Results of the average number of colours used through the phase transition.

and Cosine showed more promise than the two strategies that restricted to just using knowledge about the current vertex. In order to get a strong comparison, we compared all the strategies on a suite of generated problem instances that encompass the phase transition. This way we ensure a comparison on very hard-to-solve problems. DSatur is a clear winner of the $3-$colourable competition on 200 vertex graphs. However, in sections 10.2.9 and 10.2.12, it will turn out that this is just due to backtracking, because DSatur's performance is similar on these instances to the novel strategies. Note that backtracking can cause a problem when the problem size scales up, but the evolutionary algorithms may maintain their performance. The results confirmed that on the benchmarks, the two novel strategies are more effective, i.e. they had a higher success ratio, and fewer number of colours were used in the phase transition and its right hand side when $\chi \geq 5$. Also, they were far more efficient, and more consistent in their efficiency; that is, they had smaller deviation of the results.

10.2.8 Algorithms in the CU Merge Framework

This section provides two novel algorithms introduced by *the author* in [100]. These algorithms how the novel Dot Product and Cosine strategies (see Section 9.7 and 9.8) in the CU Merge Framework can be applied. In sections 10.2.2 and 10.2.5, the Binary Merge Table implementations and their performance were analysed. In this section we present an application of the usage of Binary Merge Square Models. However the other Merge Models can be also utilised in accordance with Chapter **??**.

Similar to the Erdős heuristic (see Section 10.1.2) which is also a CU type algorithm, the colour choice is greedy and the Dot Prod and Cosine row-pair choice strategies are applied as a second row choice strategies. Here they choose uncoloured row for a greedily selected coloured row, i.e. always on the last coloured row. This scheme is the so-called independent set approach in accordance with it traditional name (see Section 4.2.1), where the colour selection is greedy. However not just a greedy coloured row choice can be applied. Hence a clearer characterisation is given by the CU Merge Framework, where not only a greedy coloured row/colour class choice can be applied. An experimental comparison is provided below with well-known benchmark algorithms, which are described in Section 4.2.

$CU^{greedy}_{dotprod}(A$ adjacency matrix)
1 $M \leftarrow A$
2 $u \leftarrow [\,]$ // Empty choice of an uncoloured row index
3 **repeat**
4 $c \leftarrow \arg\min_i\{i : M^{col}_{ui} = 0\}$ // Choose the first available coloured row
5 **if** $c = [\,]$
6 **then** $\mathbf{r} = \mathbf{1}$
7 **else** $\mathbf{r} = \mathbf{M}_c$
8 $u \leftarrow \left[\arg\max_i \left\{ \langle \mathbf{r}, M_i\rangle : M^{col}_{ci} = 0 \right\}\right]_1$ //Choose by **max. dot prod.**
9 $M \leftarrow \mathrm{merge}(M, \{u, c\})$
10 **until** M^{unc} is empty
11 **return** M

$CU^{greedy}_{cos}(A$ adjacency matrix)
1 $M \leftarrow A$
2 $u \leftarrow [\,]$ // Empty choice of an uncoloured row index
3 **repeat**
4 $c \leftarrow \arg\min_i\{i : M^{col}_{ui} = 0\}$ // Choose the earliest available coloured row
5 **if** $c = [\,]$
6 **then** $\mathbf{r} = \mathbf{e}$
7 **else** $\mathbf{r} = \mathbf{M}_c$
8 $u \leftarrow \left[\arg\max_i \left\{ \frac{\langle \mathbf{r}, M_i\rangle}{\|\mathbf{r}\|\|M_i\|} : M^{col}_{ci} = 0 \right\}\right]_1$ //Choose col. row by **max. cosine**
9 $M \leftarrow \mathrm{merge}(M, \{u, c\})$
10 **until** M^{unc} is empty
11 **return** M

It always chooses the last coloured row index c. When it is empty, i.e. $c = [\,]$, then instead of the row $\mathbf{r} = M_c$ the e vector (the vector with all one entries) are selected. Hence the maximisation process in the uncoloured row search can be performed. Hence the Dot Product strategy takes the maximum row sum and provides the 'maximal degree vertex'. Similar to the Erdős algorithm (see Section 10.1.2), when $c = [\,]$, the merge is a simple record of the chosen uncoloured row M_u in the coloured row; that is, it

puts it into the coloured sub Merge Matrix. When $u \neq [\,]$, the coloured row choice is performed by either the maximal Dot Product strategy or Cosine strategy (see sections 9.7 and 9.8). The $\mathbf{r}$ is a constant in the cosine maximisation, so it can be left out from the expression; hence only the $\frac{\langle \mathbf{r}, M_i \rangle}{\|M_i\|}$ is considered, i.e. the length of the orthogonal projection of $\mathbf{r}$ onto M_i.

10.2.9 Experiments

Here we describe the experimental results of two Merge Algorithms which apply coloured row choice strategies introduced by *the author* in [100].

Algorithms introduced by *the author* in [100]

These algorithms were presented in Section 10.2.8.

The $CU^{greedy}_{dotprod}$ algorithm takes the last available coloured row and merges as many uncoloured rows with it as possible, using a maximum Dot Product strategy. It applies the Binary Merge Square (see Section 9.7).

The CU^{greedy}_{cos} algorithm uses the same principle as $CU^{greedy}_{dotprod}$, but the coloured row choice is based on the Cosine strategy (see Section 9.8).

Benchmark algorithms

Benchmark algorithms were implemented in a suitable Merge Framework, so as to have a common basis for a comparison. Hence their running times differ slightly. The experiments focuses on their effectiveness; that is, how many colours they used in their colouring.

$CU^{greedy}_{Erdös}$: It is based on the Integer Merge Square Model in the CU Merge Framework. It takes the last available coloured row and merges as many uncoloured rows with it as possible, using a minimum uncoloured degree strategy detailed in Section 10.1.2.

UC^{greedy}_{dsatur}: A non-backtrack version of the DSatur algorithm, it performs only one colour assignment applying the saturation degree heuristic based on the Integer Merge Table Model and UC Merge Framework (see Section 10.1.1).

Benchmark graphs

The benchmark graph set is the same $k-$chromatic ($k \in \{3, 5, 10, 20\}$) equipartite graph set in the phase transition as that in the experiments described in Section 10.2.6. But here 20 unique instances were generated per probability group, except for $k = 30$, where 30 instances were examined to get a better confidence limit here because the two algorithms had a similar performance.

Means of Comparisons

The compared algorithms perform only one colour assignment without any backtracking or other space exploration. Hence, the experiments just compared their efficiency considering single colour assignments. Therefore only one run was necessary. On each instance we performed one run. The number of colours obtained in the runs were averaged over the edge probability groups, i.e. graphs having the same same edge probabilities. The confidence intervals were also calculated, but they just confirmed our anticipated results, hence they were not plotted here.

Results

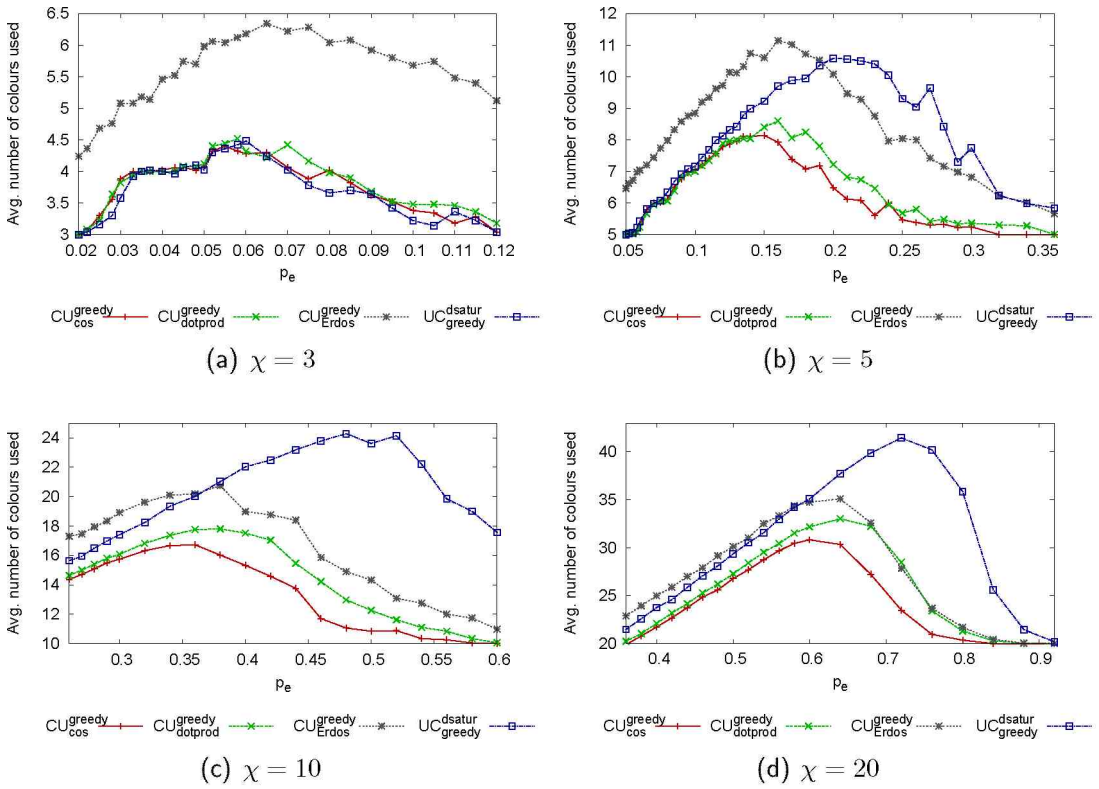
Figure 10.4 shows the results for each algorithms. The Cosine strategy performed clearly better than the others except for the $3-$colouring where DSatur performed equally well. The Dot Product strategy was ranked second, while DSatur performs well on sparse graphs having small chromatic number, the Erdős heuristic performs well on graphs that require more colours, especially on dense graphs, i.e. that have a high average number of edges (high edge density). What is interesting is the location of the region of the phase transitions. Figure 10.4 shows that it depends not just on the edge density of the graphs but also on the applied algorithm, especially the graph density where DSatur exhibits its worst performance when it moves away from the others with increasing k. DSatur and Erdős heuristics apply just second order information, as opposed to the other two algorithms, where first order information is used (see first and second co-structures in Section 7.2). The Erdős heuristic uses the secondary order structures in the opposite way to that of DSatur and our results show how this affects the performance, since their effectiveness are opposite as well.

10.2.10 Conclusions

In the UC experiments in Section 10.2.5, where Dot Product and Cosine strategies were applied for coloured row choices, here these strategies performs well too as uncoloured row choice strategies. The connection between the performance and the structure of the DSatur and the Erdős heuristic were characterised well in the Merge Frameworks. They use the same merge co-structure but in opposite way, hence their performance goes in the opposite direction when the chromatic number of the graphs in question change.

10.2.11 Algorithms in the CC Merge Framework

In the CC Merge Framework, coloured and uncoloured rows are not distinguished. A strategy always takes every row into account. Only one, a row-pair choice strategy must be defined to perform a merge sequence until a final Merge Matrix is obtained. In order to represent an algorithm in the CC Merge Framework let us introduce the following notation: $CC-\textsc{choose}$, where $\textsc{choose}$ stands for the only row-pair choice strategy.



(a) $\chi = 3$ (b) $\chi = 5$

(c) $\chi = 10$ (d) $\chi = 20$

Figure 10.4: Results of the average number of colours used through the phase transition.

The general CC Merge Framework is defined as follows with a general CHOOSE row-pair choice strategy:

CC - CHOOSE(A adjacency matrix)

1 $M \leftarrow A$
2 **repeat**
3 $\{i,j\} \leftarrow \arg\text{choose}_{\{i,j\}}\{M_i, M_j : i \neq j,\ M_{ij} = 0\}$ //Choose two row indices[a]
4 $M \leftarrow \text{merge}(M, \{i,j\})$ //Merge i and j rows/columns
5 **until** M is not mergeable
6 **return** M

[a] $M_{ij} = M_{ji} = 0$ is the merge condition, i.e. there is no edge.

Four different novel algorithms will be defined which are introduced by *the author* in [94; 97; 101; 102] and apply the four strategies of Section 9.7, 9.8, 9.6 and 9.9. Each of them is based on the Binary Merge Square Model. However the other Merge Models can be also utilised in accordance with Chapter ??. The strategies are described below. Here just the $\{i,j\} = \arg\text{choose}_{\{i,j\}}\{M_i, M_j : i \neq j\}$ general choice strategy is defined as a replaceable part of the general CC Merge Framework.

Dot Product $(CC - dotprod)$

$$\{i,j\} = \arg\max_{\{i,j\}} \{\langle M_i, M_j \rangle (1 - M_{ij}) : i \neq j\}$$

Two rows are chosen for a merge if they have maximal dot products among the possible row pairs. The $M_{ij} = 0$ merge condition can be given by using the $(1 - M_{ij})$ term in the case of Binary Merge Squares.

Cosine $(CC - cos)$

$$\{i,j\} = \arg\max_{\{i,j\}} \left\{ \frac{\langle M_i, M_j \rangle}{\|M_i\| \|M_j\|} : i \neq j,\ M_{ij} = 0 \right\}$$

Two rows are chosen for a merge if they have a maximal cosine among the possible row pairs.

Approximated spectral norm $(CC - \tilde{\sigma})$

$$\{i,j\} = \arg\min_{\{i,j\}} \left\{ \sqrt{\sum_{r=1}^{l} \langle (M_{/ij})_r, (M_{/ij})_r \rangle^2} : i \neq j,\ M_{ij} = 0 \right\}$$

Two rows are chosen for a merge if they have a minimal approximated spectral norm among the possible row pairs. $M_{/ij}$ is the Merge Square after merging i and j rows, where $(M_{/ij})_r$ is the r-th row of the merged matrix and l is the number of rows/columns in the merged matrix. This definition follows from Eq. 9.27, where $\langle (M_{/ij})_r, (M_{/ij})_r \rangle = \langle (M_{/ij})_r, \mathbf{e} \rangle$ is the r-th row sum, due to the Binary Merge Matrix representation, and the constant term l is left out of the denominator. This strategy can be defined without $M_{/ij}$ trial merges by an efficient direct calculation and an update technique (see sections 9.6 and 9.6 for details).

Zykov-tree+Lovász-theta $(CC{-}Zykov_{\bar{\theta}})$

$$\min_t \{t : Z \succeq 0, z_{ii} = t - 1, z_e = -1\ \forall e \in E\}$$

Let the approximated solution of this semi-definite optimisation problem be $\tilde{Z}_{opt}$ in accordance with Eq. 9.43. Two solvers were applied for this optimisation. In order to get a faster execution the combination of a boundary point method [134] and an interior point method [142] is applied. Later it was applied for very dense graphs (edge density>0.89) [4], when the candidate solution approached, the final Merge Matrix. Otherwise the calculation was done by a boundary point method. The row-pair choice strategy for a merge was defined by the and $\hat{Z} = (\tilde{Z}_{opt} + 1) \circ (1 - I)$ (see Section 9.9) as follows:

$$\{i,j\} = \arg\max_{\{i,j\}} \left\{ \hat{Z}_{ij}(1 - M_{ij}) : i \neq j \right\}$$

In order to further improve the speed and the decision accuracy, an (i, j) edge addition was introduced for each step using the following

$$\{i, j\} = \arg\min_{\{i,j\}} \left\{ \hat{Z}_{ij}(1 - M_{ij}) : \hat{Z}_{ij} < 0 \right\}$$

Further details can be found in Section 9.9. $CC - Zykov^{+}_{\bar{\theta}}$ will stand for the variant when not only one edge, but all edges are added which satisfy the following condition[5]

$$\left\{ \hat{Z}_{ij}(1 - M_{ij}) : \hat{Z}_{ij} < 0 \right\}$$

10.2.12 Experiments

The experimental setup was the same as that outlined in Section 10.2.9.

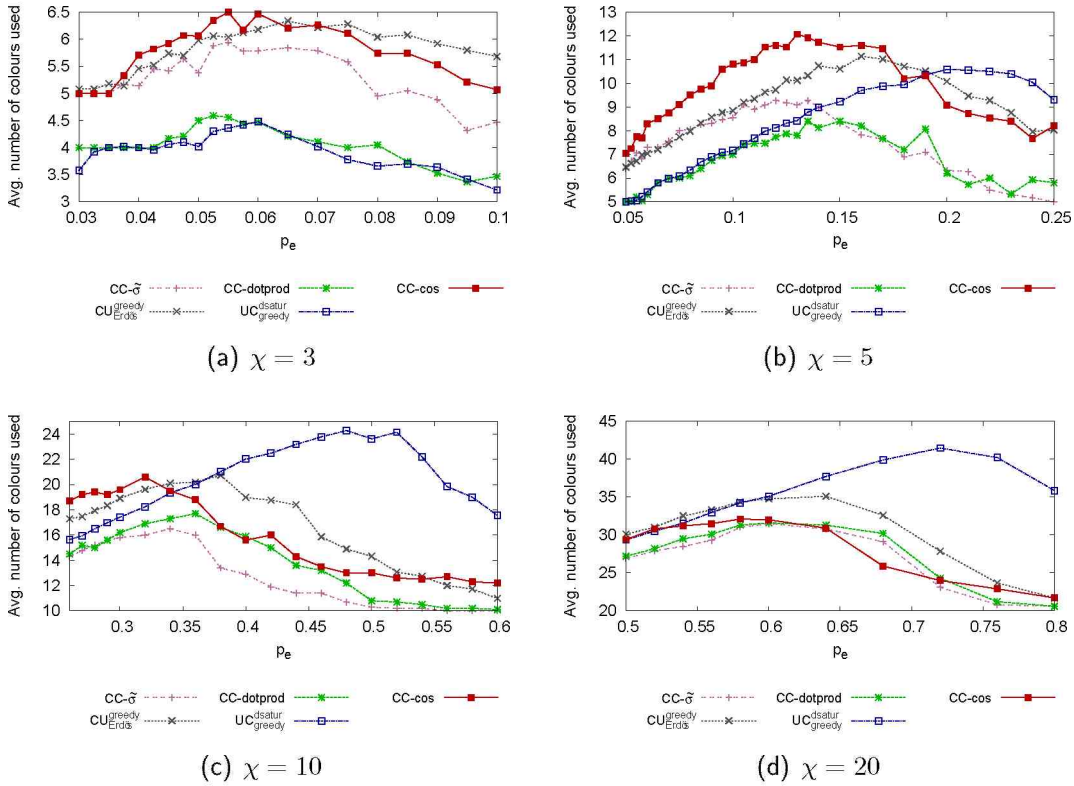
Results

Figure 10.5 shows the results of every combination for different values of χ. Hence the spectral norm approximation performs the best except for very sparse graphs, when the Dot Product strategy and DSatur with local decisions perform better. The reason for the worse performance of $CC - \tilde{\sigma}$ on sparse graphs is the small number of changes in the norm in the selection of candidate vertices pairs for a merge. Because of the approximation used, several different values become the same, hence too many candidates are selected for tie breaking. The combination of the CC framework with the Cosine does not always perform well, especially for smaller chromatic numbers; however, it can outperform baseline methods for dense graphs. As the chromatic number and the edge density increase Cosine strategy increases its performance and it can beat every other. Dot Product's performance lies between that of Cos and the $CC - \tilde{\sigma}$ algorithms; its strength lies with smaller chromatic numbers and sparse graphs. Figure 10.6 gives the best and the benchmark results of Figure 10.5. Furthermore, Figure 10.6 shows the $(CC - Zykov_{\bar{\theta}})$ and $(CC - Zykov^{+}_{\bar{\theta}})$ results as well. Both the novel $(CC - Zykov_{\bar{\theta}})$ strategy and the $(CC - Zykov^{+}_{\bar{\theta}})$ strategy perform very well especially for denser graphs. The phase transition is shifted for these algorithms, where other algorithms can outperform their impressive results. Where more edges are added in the $(CC - Zykov^{+}_{\bar{\theta}})$ strategy it has slight influence on the results of the $3-$chromatic experiments, but its performance worsen in the higher chromatic region.

10.2.13 Conclusions

All the novel strategies presented here perform well in the CC Merge Frameworks, and most cases they outperform the benchmark algorithms. Though there is no clear winner, the $(CC - Zykov_{\bar{\theta}})$ algorithms achieve quite impressive results, but the other algorithms can outperform their results in their phase transition region. Furthermore,

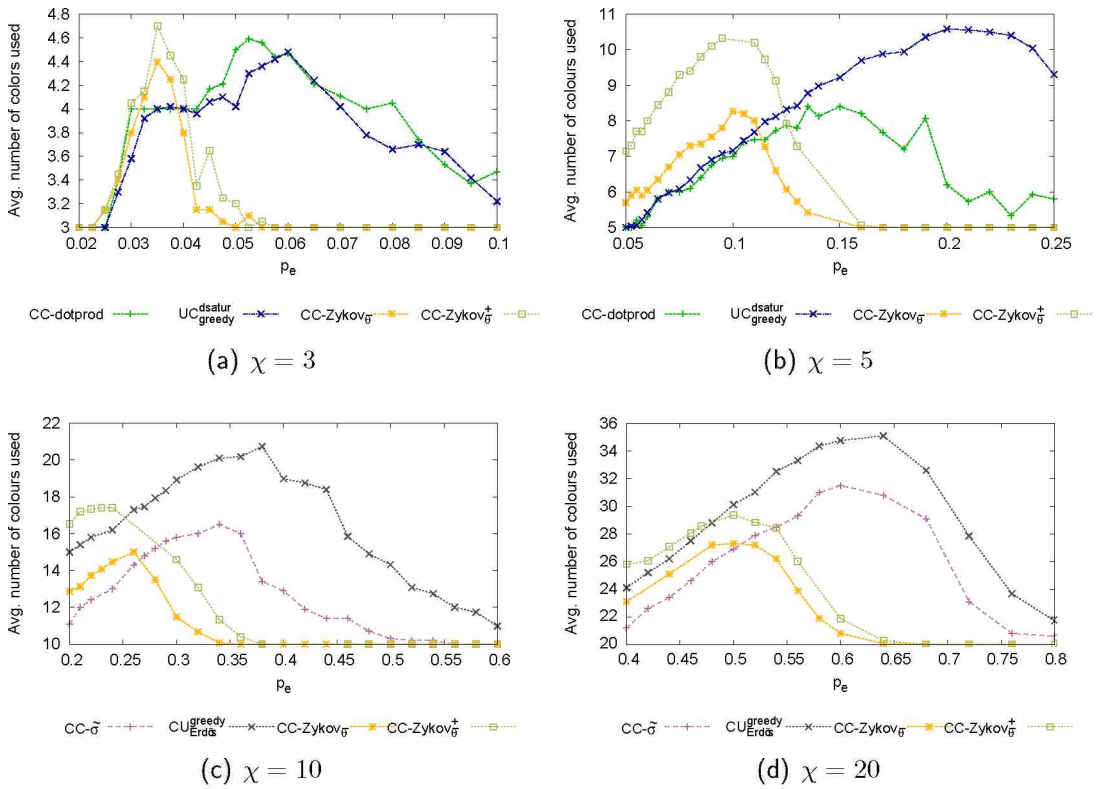
[5] Not only that edge, which corresponds to the minimum value.



(a) $\chi = 3$ (b) $\chi = 5$

(c) $\chi = 10$ (d) $\chi = 20$

Figure 10.5: Results of the average number of colours used through the phase transition.



(a) $\chi = 3$ (b) $\chi = 5$

(c) $\chi = 10$ (d) $\chi = 20$

Figure 10.6: Results of the average number of colours used through the phase transition.

the $(CC-\tilde{\sigma})$ has also good results especially for graphs which have a higher chromatic number. Nevertheless, $(CC - Zykov_{\bar{\theta}})$ colours efficiently these graphs and they use much more computational effort than the others. They have to perform several semi-definite optimisations to achieve these good results. These optimisations make them slower than the others, which use only a couple of elementary operations for their strategies, hence they are suitable for solving larger graph instances.

10.3 Summary

In this chapter we demonstrated the efficiency of the strategies described in Chapter 9 when embedded into one of the Merge Frameworks. Our experiments showed that they perform well when applied as uncoloured or coloured row choice strategies, or row-pair choice strategies. Benchmark algorithms were defined in a suitable Merge Framework, and these definitions allowed us to make a structural comparison. In addition, our experimental results and the structural analysis revealed a correlation in the case of DSatur and Erdős heuristics.

In the next section we will look at the Merge Models and Algorithms in more detail.

Chapter 11

Analysis

11.1 Introduction

In chapters 7, 8 and 9 a new general colouring approach was constructed based on a special graph homomorphism. Chapter 10 showed the practical benefits of these approaches by an experimental investigation. This chapter shows the result of a theoretical analysis of the approaches and discusses software and hardware implementation aspects, as described by the author *in [94; 96–102].*

11.2 Which Merge Model is better?

This section briefly discusses the benefits and drawbacks of the Merge Models introduced in Chapter 7.

Integer or Binary Merge Matrices Integer Merge Matrices supports backtracking because they retain all the edges of the original graph, and hence a merge is reversible. Instead of row addition, a row subtraction can provide an unmerge operation, supporting backtracking. In Chapter 10 we presented algorithms which exploit the co-structure properties of these models, such as the Erdős and DSatur heuristic use them for tie breaking. Furthermore, they support the extension of the Merge Models for improper colouring schemes, as outlined in Section 10.1.1. However, retaining all the edges can support some strategies, but there may be drawbacks as well. The edge retaining introduces redundancies hence the final Integer Merge Matrix is not predictable. The final Merge Matrix is defined exactly for the Binary Merge Squares it is the adjacency matrix of the K_k complete graph. This is useful in algorithm design (see Section 9.10). Since a Merge Square is an adjacency matrix, strategies can always be repeated in these merged matrices, but several strategies can work with the other merge matrices as well. Unfortunately, it does not support backtracking, as it loses some of the original edges. An implementation of a Binary Merge Matrix can be effective because the binary operations and structures are supported by the computer hardware and software.

Merge Squares or Merge Tables The implementation of the Merge Tables can be done more efficiently, since the merges affects only the rows, while in the case of Merge Square the two dimensional changing, i.e. the number of rows and columns, requires

more computation and cause implementation problems. Nevertheless, the Merge Square structure better supports an analysis, hence their structure and the dimensions are similar to the original problem and are described by similar graphs. But a Merge Table always makes available the original graph structures, as their columns represents the vertices of the original graphs.

11.3 Enhanced algorithms

In sections 9.4.1 and 9.4.2 two novel algorithms of *the author* were introduced based on two current non-merge based algorithms, namely the Welsh-Powell heuristic and the Hajnal heuristic (see Section 4.2.3 and 4.2.4). The extended version of these in the UC Merge Framework brings an improvement in the performance of the original algorithms (see Juhos et al. [97]). Sections 9.4.1 and 9.4.2 detailed the improvements in a theoretical point of view and here several experimental results demonstrate our findings in a standard benchmark set of graphs (see Section 4.1). Table 11.1 clearly shows the difference in performance between the original version and the extended version of the algorithms. The bias was set to $\nu = 0.9$ in each experiment performed (see Eq. 9.4). The novel extended version of the algorithms clearly outperforms the original

Graph	$\|V\|$	$\|E\|$	χ	UC_{greedy}^{WP}	UC_{greedy}^{ext-WP}	UC_{greedy}^{H}	UC_{greedy}^{ext-H}
queen10_10	100	2940	?	17	15	16	14
queen11_11	121	3960	?	17	15	20	15
queen12_12	144	5192	?	19	17	22	17
queen13_13	169	6656	13	23	18	23	18
DSJC125.1	125	1472	?	7	6	9	7
DSJC125.5	125	7782	?	23	22	26	22
DSJC125.9	125	13922	?	53	52	62	54
DSJC250.1	250	6436	?	11	10	14	11
DSJC250.9	250	31366	?	93	88	97	89
DSJC500.1	500	24916	?	18	16	22	16
DSJC500.5	500	125249	?	71	66	76	67
DSJC500.9	500	125249	?	169	165	185	166
flat300_20_0	300	21375	20	44	42	46	43
flat300_26_0	300	21633	26	47	44	50	43
flat300_28_0	300	21695	28	44	43	48	44
latin_square_10	900	307350	?	213	148	148	145
le450_5c	450	9803	5	12	8	19	10
le450_5d	450	9757	5	14	9	19	11
le450_15a	450	8168	15	18	17	26	18
le450_15c	450	16680	15	26	25	36	25
le450_25a	450	8260	25	26	25	34	25

Table 11.1: Results of extended algorithms. The number of colours used by the Welsh-Powell (WP) and the Hajnal (H) algorithms and their extensions. The extended algorithms are denoted by 'ext-' prefixes.

two algorithms. These experiments are based on the graphs of the DIMACS benchmark repository (see Section 4.1). The experiments were performed on the same graph set as those applied in the experiments described in Section 10.2.2. The extended algorithms produced much the same results, outperforming the original ones. Here some other difficult-to-solve instances are presented in order to demonstrate the efficiency of the novel algorithms.

11.4 Reduced computational cost

In [52; 145], Eiben and van Hemert et al. pointed out that the number of constraint checks is the key factor in the computational cost in colouring algorithms. However, there can be other factors which affect the running time; constraint checks characterise well the computational efforts several times. Merge Models provide considerable decrease in running time for those algorithms which performance strongly correlated with the constraint checking (see Juhos et al. [99]). This section introduce the results of *the author*. Our benchmark algorithms in Section 4.2 are typical examples for such graph colouring solvers. When solving a graph colouring problem as a sequential colouring while using the original graph representation to check for violations, approximately n^2 $(= |V|^2)$ constraint checks are required to get to a valid colouring. In contrast, a Merge Model (MM) supported scheme uses at most $n \cdot k$ number of checks ($|V| \geq k \geq^2 \chi$). This is possible because each vertex will be compared with at most the existing colour classes, of which there are no more than k or χ if a solution exists. Hence, their quotient determines the improvement of a Merge Model supported colouring, which is proportional to the n/k ratio. We verify this claim theoretically and empirically as well. In traditional schemes, adjacency matrix representation plays the key role in the GCP [1]. We have two choices when colouring a vertex for constraint checking; either along the already coloured vertices ($\mathcal{A}_{col}$), or along all the neighbours of the vertex considered ($\mathcal{A}_{neigh}$). In the following, we show how to considerably reduce the number of constraint checks by applying our proposed Merge Models ($\mathcal{A}_{mm}$). Let π is the sequence of the vertices occur in the colouring process. Define $\overset{\frown}{d}(x)$ as the coloured-degree of the vertex x being currently coloured, which refers backwards to the already coloured vertices and $\overset{\frown}{d}(x)$ of the uncoloured-degree refers forwards to the uncoloured vertex. Furthermore, denote $k_{\pi(x)}$ the number of colours used before x would have been coloured according to π. Notation $\sum$ is always $\sum_{i=1}^{n}$ in this section.

Corollary 11.1 ([99]) *Given a random graph $G_{n,p}$ with fixed p edge probability and given a colouring algorithm $\mathcal{A}$, then the following performance is expected on average based on counting constraint checks $\#(.)$:*

1. *checking the coloured vertices:* $\#(\mathcal{A}_{col}) = \mathcal{O}(n^2)$
2. *checking the neighbours:* $\#(\mathcal{A}_{neigh}) = \mathcal{O}(n^2)$
3. *checking the merged-vertices/colour classes:* $\#(\mathcal{A}_{mm}) \leq \mathcal{O}\left(\frac{n^2}{\log n}\right)$

Proof

1. Checking the already coloured vertices requires as many neighbour checks as the number of the edges, because we have to check the t number of coloured vertices if the $t+1$. vertex comes to colour, that is,

$$\#(\mathcal{A}_{col}) = \sum i = \frac{1}{2}n(n-1) = \mathcal{O}(n^2) \tag{11.1}$$

2. When the neighbours of the vertex currently being coloured are checked for constraint violation, the number of performed constraint checks are equal to the sum of the degrees, i.e., twice the number of edges

$$\#(\mathcal{A}_{neigh}) = \sum d_i = 2|E| \propto pn(n-1) = \mathcal{O}(n^2) \tag{11.2}$$

3. Using a Merge Model representation, Merge Operations provide merged-vertices, which represent colour classes, thus checking along them, requires at most as many checks as the number of colours used at that moment. The worst case is when the colouring is tight, meaning vertex x is in position $\pi(x)$ coloured by at least the colour $k_{\pi(x)}$.

$$\begin{aligned} \#(\mathcal{A}_{mm}) \leq \sum k_i = & \quad n\tfrac{\sum k_i}{n} \propto nr\chi \propto n\tfrac{rn}{2\log n} \\ = & \quad \mathcal{O}(n^2/\log n) \end{aligned} \tag{11.3}$$

$$n\frac{rn}{2\log n} \propto \frac{rp}{2p}\frac{n(n-1)}{\log(n-1)} = \frac{pn(n-1)}{\log(n-1)^{2p/r}} \tag{11.4}$$

where r is constant[2] and $\chi \approx \frac{n}{2\log n}$ according[3] to [13]. For further details see Section 3.7 and 4.2.2. □

As the theorem above tells us the asymptotic behaviour of the algorithms, we can check the worst case behaviour of the $\mathcal{A}$ using these different approaches. It is clear that the application of the $\mathcal{A}_{col}$ for dense graphs are better against the $\#(\mathcal{A}_{neigh})$, and conversely, $\mathcal{A}_{col}$ has worse properties in sparse graphs compared to $\#(\mathcal{A}_{neigh})$. The following theorem states that using our Merge Model approach, $\#(\mathcal{A}_{mm})$ will always outperform the other techniques mentioned previously.

Corollary 11.2 ([99]) *Let G be an arbitrary graph, then the following relations hold*

1. $\#(\mathcal{A}_{mm}) \leq \#(\mathcal{A}_{col})$
2. $\#(\mathcal{A}_{mm}) \leq \#(\mathcal{A}_{neigh})$

Proof

1. The number of colours is less than the number of coloured vertices, i.e. $\#(\mathcal{A}_{mm}(x)) \leq k_{\pi(x)} \leq \pi(x)$, and $\#(\mathcal{A}_{mm}) \leq \sum k_i \leq \sum i$.

[2] This is true in the context of the naturally defined greedy algorithms $r \approx 2$ [44; 71; 117], but other algorithms have been designed to perform better.

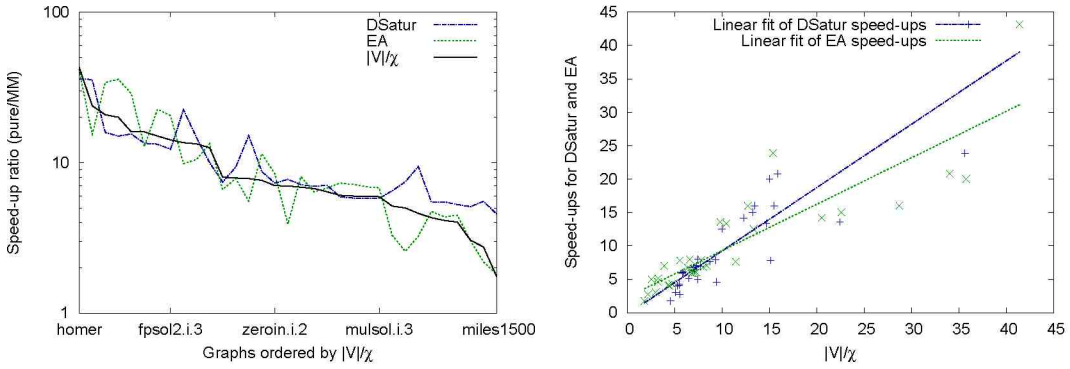
[3] Note that the logarithmic base here is $\frac{1}{1-p}$.

2. If $\overset{\curvearrowleft}{d}(x)$ refers to distinctly coloured vertices then $\#(\mathcal{A}_{mm}(x)) = \overset{\curvearrowleft}{d}(x)$. Otherwise, if $\overset{\curvearrowleft}{d}(x)$ refers to the same coloured vertices as well as distinct ones then $\#(\mathcal{A}_{mm}(x)) = \overset{\curvearrowleft}{d}(x)$, since merged-vertices encompass the same coloured vertices. Consequently,

   $\#(\mathcal{A}_{mm}(x)) \leq d(x)$ due to $d(x) = \overset{\curvearrowleft}{d}(x) + \overset{\curvearrowright}{d}(x)$, and
   $\#(\mathcal{A}_{mm}) \leq \sum \overset{\curvearrowleft}{d}(x) \leq \sum d(x)$. □

One consequence of Corollary 11.2 tells us more. Namely, an MM based algorithm could perform better than that which could just check the coloured neighbours. However, to implement such an algorithm, which just checks the coloured neighbours, we have to use additional computation efforts. Thus an MM algorithm performs even better.

In Figure 11.1(a), we show how much the speed of DSatur and the evolutionary algorithm (see Section 10.1.1) increases when measured as the ratio of constraint checks used to solve the problem when without using Merge Models and when using Merge Models on a standard test benchmark set of graphs (see DIMACS problems in Section 4.1). For DSatur, the lowest speed increase is 4.56, while the largest speed-up is 36.1. For the evolutionary algorithm, the lowest speed increase is 1.81, and the largest speed-up is 41.4. Depicted in Figure 11.1(b) is the correlation of the speed-up ratios of the two algorithms with the ratio n/χ, i.e., the number of vertices divided by the chromatic number. DSatur has a constant of proportionality of 0.948 and an asymptotic error of 10.0%, while the evolutionary algorithm has a constant of proportionality of 0.695, and an asymptotic error of 9.8%. We predicted this speed-up in Section 11.4, and our theory agrees well with the observed correlation, especially for DSatur.



(a) Speed increase ratios which are ordered by $|V|/\chi$ in decreasing order.

(b) The linear correlation seen between the $|V|/\chi$ ratio and the speed-up ratios.

Figure 11.1: Speed increase of DSatur and the evolutionary algorithm (EA) for the DIMACS problems after embedding them into a Merge Model (MM). 'Pure' corresponds with the algorithms without using a Merge Model.

11.5 Implementations

In a k-colouring, we need just $|V| - k$ contraction steps to get a solution instead of the n required by the traditional colouring methods, e.g. in a colour assignment. A lot of hardware nowadays provides CPUs with vector operations, which opens up the possibility of performing the atomic Merge Operations in one CPU operation, thereby raising the overall efficiency. Since nowadays computer processing units (CPUs) support parallel operations, e.g. vector addition operations (VADD) or vector OR operations (VOR). Hence, a Merge Operation may be only one instruction instead of $n = |V|$ or $d(x_i)$ instructions. In this case at most $n - k$ number of VADD or VOR operations are needed for a valid colouring. The order of real-life graphs can vary from a hundred vertices to thousands of vertices. Using special hardware instructions available on modern computers, Merge Operations can be reduced to one computer instruction. For example, a Merge Operation can be performed as one VADD or VOR operation on a vector machine, such as the Xbox game station [32]. The IBM PowerPC CPU used in an Xbox [32] has $49\,152$ $(3 \cdot 128 \cdot 128)$ bits for this operation. Thus we can use one binary Merge Operation for graphs having at most $49\,152$ or one integer Merge Operation for graphs having at most $4\,000$ number of vertices. The latter is due to the fact, a cell value of an Integer Merge Matrix is always being less than n. Hence in the Integer case the n is calculated by $n\lceil \log_2 n \rceil = 49\,152$, because $\lceil \log_2 n \rceil$ bits are required for each of the n integer-valued cell of a row. Note that in the case of Merge Squares the dimension of the rows decreases, hence after a certain number of merges the further Merge Operations will require only one VADD or VOR operation. Nevertheless, having a smaller VADD of VOR size, say l, the necessary VADD or VOR operations are $\lceil n/l \rceil (n - k)$, which can still significantly reduce the computational efforts for a merge. In particular, if $l \geq n$, then we get back the $n - k$ as mentioned previously. Examples that show how such hardware can speed up computation can be found in surveys in [34; 141]. In recent years, we have witnessed a surge in low-cost hardware that is capable of efficiently performing specific operations. An important reason behind this surge is the extensive use of Graphics Processing Units (GPU) in computer games and recently, in computer consoles. This in turn has led to general-purpose computation on GPUs, which can provide a number of advantages over traditional high-performance computing facilities. Specifically, in the context of General-Purpose computation on Graphics Processing Units (GPGPU), the advantage is that accelerated graphics cards are now cheap, and most desktop and laptop machines contain one with a large number of GPUs. Their energy consumption is considerably lower than that of Central Processing Units (CPUs). By making use of them in a computational sense, we bring parallel computing hardware as close to the end-user as we wish. We can compute on their desktop or provide remote access to GPUs installed elsewhere. Also, recent advancements in speed seem to be in favour of the GPU, not the CPU [38]. Besides hardware implementations, parallelism can be achieved via a software implementation as well [3]. Parallel computing on one machine or distributed computing on several machines may be also options for a software implementation. In analogy to the hardware implementations, a Merge Operation can be distributed

using an appropriate software package which can support either parallel or distributed computation. However, software packages may have computational overheads but for extremely large graphs their usage can be worthwhile.

11.6 Summary

In this chapter we analysed the application of the Merge Models from various aspects. We demonstrated improvements in the performance of an algorithm after embedding it into a suitable Merge Model. Without any change in the algorithm steps, the representation of the problem in a Merge Model provides a decrease in the computational effort. However, the embedding permits a natural enhancement of the algorithm as well. It may bring significant improvements in the performance of an existing algorithm. After, practical implementations issues were discused, which can further improve the efficiency of a concrete implementation of a Merge Model on a particular hardware or software platform.

Appendix

11.7 Symmetry in the colour assignment

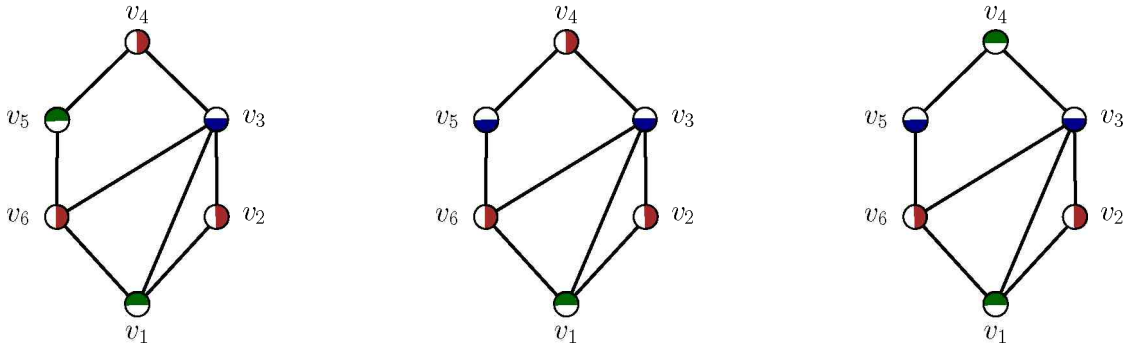
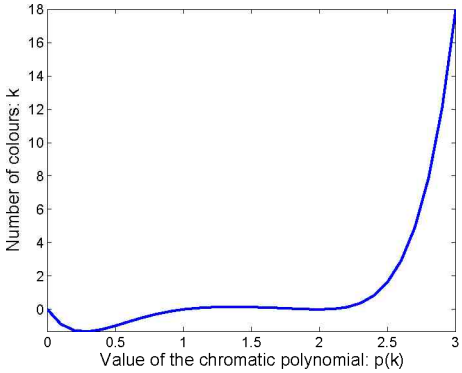


Figure 11.2: Different optimal colourings for the graph shown in Figure 3.1

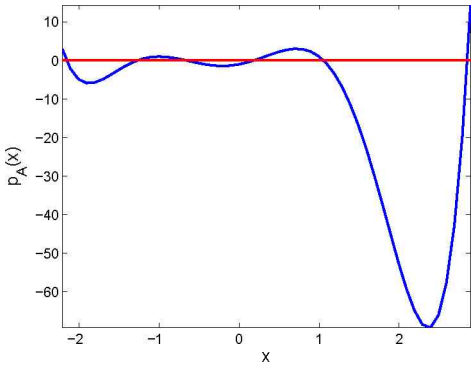
Serial	v_1	v_2	v_3	v_4	v_5	v_6
No. 1:	1	2	3	1	3	2
No. 2:	1	2	3	2	1	2
No. 3:	1	2	3	2	3	2
No. 4:	1	3	2	1	2	3
No. 5:	1	3	2	3	1	3
No. 6:	1	3	2	3	2	3
No. 7:	2	1	3	1	2	1
No. 8:	2	1	3	1	3	1
No. 9:	2	1	3	2	3	1
No. 10:	2	3	1	2	1	3
No. 11:	2	3	1	3	1	3
No. 12:	2	3	1	3	2	3
No. 13:	3	1	2	1	2	1
No. 14:	3	1	2	1	3	1
No. 15:	3	1	2	3	2	1
No. 16:	3	2	1	2	1	2
No. 17:	3	2	1	2	3	2
No. 18:	3	2	1	3	1	2

Table 11.2: The all optimal colourings for the graph shown in Figure 3.1. Including equivalent colourings, which cause symmetric solutions. A serial number shows the appropriate colouring. Numbers below the header v_i—s are the colour assignments e.g. $\mathbf{No.1}$: $c(v_1) = 1$, $c(v_2) = 2$, $c(v_3) = 3$, $c(v_4) = 1$, $c(v_5) = 3$, $c(v_6) = 2$ where $c(v)$ is the colour identifier assigned to the vertex v. Some solutions generate the same solutions. E.g. $\mathbf{No.1}$ colouring is equivalent with $\mathbf{No.4}$.

11.8 Characteristic and chromatic polynomials



(a) A plot of the values of the chromatic polynomial: $p(k) = k^5 - 8 + 26k^4 - 43k^3 + 36k^2 - 12k$. Where $p(3) = 18$ (see Table 11.2).



(b) A plot of the values of the characteristic polynomial: $x^6 + 0x^5 - 8x^4 - 4x^3 + 9x^2 + 4x^1 - 1$.

Figure 11.3: The plots of different polynomials of Figure 3.1

Summary

This thesis summarises the results obtained by *the author* over the past few years. *The author* developed a general framework for graph colouring methods, where the traditional colouring scheme is defined via special graph homomorphisms motivated by the Zykov theorem [161; 162]. These special homomorphisms proved useful in the design of algorithms by *the author* ([94; 96–102]). This summary is structured in a similar way to the thesis itself. The results can be separated into different groups according to the parts of the graph colouring framework. *The author* defined the problem via certain graph homomorphisms using quotient and power graphs. *The author* called these *Quotient and Power methods*. Then he described these graphs and homomorphisms by matrix representations with suitable operations, resulting in his *Merge Models* with his nomenclature. Merge Models provide a novel description of the colouring problem. The operations (i.e. the *Merge Operations*) subsequently change the state of the model and direct it to a possible solution of the original graph colouring problem. *The author* developed strategies in the model called *Merge Strategies* which define possible directions to a solution. Furthermore, *the author* constructed general frameworks (*Merge Frameworks*) in which strategies can be embedded. These frameworks in conjunction with the strategies form colouring algorithms (*Merge Algorithms*). Such algorithms generate a sequence of model operations according to the strategy. The end of the sequence is a candidate solution for the original problem.

Quotient and Power Methods

The author defined graph colouring processes as a series of homomorphisms using quotient or power graphs and multigraphs, where the vertices which get the same colour will be 'glued' or 'grouped' together to form new vertex sets (see Juhos et al. [96; 100]). *The author* called the new colouring methods which are based on these principles Quotient and Power methods. The goal of a Quotient and Power method is to find a homomorphism which maps the original graph into a complete graph or homomorphic with a complete graph. The homomorphism obtained defines a colouring of the original graph. In order to support the design of sequential colouring algorithms such a homomorphism is created as a composition of series of intermediate homomorphisms. These homomorphisms produce helpful intermediate graph structures which can be exploited for an efficient colouring and also help provide a deeper insight into the colouring procedure. Moreover, they allow us to design efficient new or redesign existing graph colouring algorithms in a framework supported by quotient or power graphs (see Juhos et al. [96–102]).

Merge Models

The relation between the original graph [4] and a quotient or power graph/multigraph is defined by a graph homomorphism. *The author* introduced four kinds of matrix operations, called Merge Operations (or 'merges' for short) to map the adjacency matrix of the original graph to its four different homomorphic images: called Binary/Integer Merge Square/Table Matrices or put briefly Merge Matrices), respectively, and then subsequent Merge Operations will produce vertex colouring [96; 100]. *The author* showed that Merge Operations produce appropriate homomorphic images of the original problem, modelling the original graph colouring problem. Each row of a Merge Matrix corresponds with an independent set in the original graph. Note that colour classes are independent sets, and each vertex constitutes a one-element independent set in the original graph. All the models have their own strong points, and they can assist each other indifferent ways. *The author* obtained significant improvements both theoretically and via experiments when an algorithm applied one of these models [99]. Exploiting their good performance, *the author* designed powerful graph colouring algorithms in [94; 97–99; 101; 102].

Merge Frameworks

Merge Models provide a model for the graph colouring problem via matrix representations and operations. *The author* introduced three general frameworks for graph colouring algorithms supported by Merge Models in [100; 101]. These are generalisations of the traditional sequential colouring schemes. Merge Models replace the colour assignment operation with a Merge Operation, and this eliminates the difference between the colour selection and the vertex selection strategies. Merge Models define these different selection strategies in a common way as a common row selection strategy. Therefore, a general row selection strategy can operate as a coloured or uncoloured row selection when we would like to model the traditional selection strategies. Here the colours only indicate whether a row has already been taken into account in the merge process. Depending on the order of the selection of the different state (coloured/uncoloured) rows two general framework can be defined: either we choose an **u**ncoloured row first and then choose a suitable **c**oloured one (UC Merge Framework) or, conversely, we can choose a **c**oloured first and then find an appropriate **u**ncoloured row for the merge (CU Merge Framework) [100]. The UC and the CU frameworks provide a generalisation of the sequential colouring schemes. In fact there is no need to distinguish between the coloured or uncoloured states of the rows; just take the set of rows and apply a common choose strategy suitable for all of them. After, select an arbitrary row-pair from the Merge Matrix by a strategy and merge them. This approach is formulated in the CC Merge Framework [96]. The rows of the Merge Matrix correspond to colour classes, i.e. independent sets. An algorithm in a CC Merge Framework selects two colour classes/independent sets and creates the union of them in the traditional sense. These

[4] Or an equivalent reformulation of the original graph.

general frameworks with the new Merge Models support a common structural analysis of the existing and novel graph colouring methods, as shown in [97; 99; 101; 102]. All of these frameworks are defined in a unified manner using the Merge Model scheme. An algorithm in one of these frameworks applies a subsequent selection of rows of the merge matrices and merges them to achieve a colouring. None of these frameworks has a concrete strategy for the choice of rows for merging. A framework with a concrete choice strategy, i.e. Merge Strategy, forms a particular algorithm.

Merge Strategies

In order to get a colouring algorithm, the algorithm steps must be defined; that is, a sequence of the Merge Operations. A Merge Operation takes two rows/columns of a Merge Matrix and produces a new Merge Matrix if the merge condition allows it. By repeating Merge Operations we will end up with a final Merge Matrix where a Merge Operation is no longer possible. The sequence of the Merge Operations is crucial. It determines the quality of the solution, i.e. the number of colours used in the colouring of the original graph. *The author* described various Merge Strategies in order to generate efficient merge sequences, as described in [94; 96–102]. These strategies proved useful in the theoretical and experimental parts of our analysis. The novel description of the colouring process provides new aspects which can be exploited in the design and analysis of Merge Strategies, as described in the following. This strategies assume Binary Merge Models, but their integer extensions are also available. The importance of the Integer Models are discussed separately. They support the algorithm design, e.g. backtracking or tie breaking, as shown in [99].

The longest merge sequence. Since the Merge Matrix rows correspond to colour classes, the main aim is to reduce the number of rows by consecutive merges. The longest merge sequence produces the fewest rows. *The author* in [97] introduced two novel strategies to generate the longest merge sequence. The Dot Product Strategy focuses on the evolution of the number of non-zero elements during successive merges and attempts to keep them as low as possible. Though the non-zero elements in a Merge Matrix frustrate the merges, the number of zeros assist them. Hence the Cosine Strategy takes the number of non-zero elements into account, but also considers the number of zeros present.

Parallel rows. The Cosine strategy favours the parallel rows in the Merge Matrices. It is reasonable because the rows of the adjacency matrix which correspond to the same coloured vertices in an optimal solution are almost parallel. Their parallel behaviour becomes clearer with each successive merge. In the case of Merge Square Model, there is a certain modification of the Merge Matrices based on a semi-definite optimisation by Karger et al. [103], which further supports the Cosine strategy. Exploiting this fact, *the author* in [94; 102] defined the Zykov-tree and Lovász-theta strategy.

Colour similarities In fact a Zykov-tree and Lovász-theta strategy is based on the estimation of the colour similarities of the vertices of the quotient graphs. The adjacency matrix describes an exact colour dissimilarity relation, where the vertices in

(edge-)relation cannot get the same colour. The opposite approach is the colour similarity relation. A particular colouring can be defined via a colour similarity relation between the vertices, where only the same coloured vertices are included in the relation. This relation can be represented by a $\{0,1\}$-matrix, namely a colouring matrix. It describes whether two vertices are coloured with the same or different colours. Although the optimal solutions can be represented in this form, they are unknown because they are the solutions of the problem. Despite this, their average can be approximated by a solution of a semi-definite program (see Karger et al. [103]), which provides the Lovász-theta. Hence, a non-exact, an approximated colour similarity relation becomes available between the vertices. This can be described by a real-valued matrix, where the largest and the smallest values contain valuable information. Using this information and Zykov's work in [161; 162], *the author* created the Zykov-tree and Lovász-theta strategy in [94; 102], where quotient graph vertices are connected or merged according to their approximated similarities. The approximation becomes more exact with each successive merge supporting more confident decisions of this strategy.

Norm minimisation in the resulting state. The Dot Product Strategy selects two rows which produce the maximum dot product, then merges them. This introduces a minimisation in the entrywise norms in the resulting Merge Matrix. A final Merge Matrix which corresponds to an optimal solution has the smallest entrywise norm among the possible merge matrices (homomorphic images). Hence, the entrywise norm minimisation approach is reasonable. In addition such a Merge Matrix has minimal induced norms as well. This observation led us to apply the steepest descent norm minimisation strategy, in particular the steepest descent Spectral Norm Strategy, which was introduced by *the author* in [101] and was found to be an efficient strategy.

The Spectral Norm Strategy must first make several trial merges. With the resulting trial merge matrices, this strategy makes spectral norm calculations to create a selection of a row-pair for merging. Calculating the spectral norm is computationally expensive, but Merikoski and Kumar once introduced an efficient spectral norm approximation in [123]. Based on their results, *the author* adapted his Spectral Norm Strategy to an approximated spectral norm strategy [101]. Owing to this, this strategy can exploit an update mechanism where an investigation of the resulting Merge Matrices is no longer needed as it is just based on the current Merge Matrix. In addition this reformulation revealed a connection with the Dot Product strategy.

Matrix properties – Merge Paths The author introduced the notion of Merge Paths [101]. Certain graph properties like matrix norms may be evaluated during the selection of two rows for a Merge Operation. Gathering these graph properties into a vector (e.g. eigenvalues) they form the basis of the decision. The changes of the property vector during the merge process describe a path called the Merge Path. This path is responsible for determining the colouring and the end of the path defines the quality of the colouring.

Unfortunately, the ideal path (which results in an optimal solution) is of course unknown; the task of colouring is to find this path. *The author* introduced a general strategy which approximates an optimal Merge Path [101]. The start and the end

points of the path are usually known and the curve of the path may be estimated by using preliminary knowledge. In order to build the knowledge base the Merge Path approach can be combined with artificial intelligence methods, such as the instance based learning or clustering in accordance with the results described in [95].

Enhanced heuristics and meta-heuristics A non-merge based colour strategy can be extended and enhanced by reformulating the strategy in a Merge Model. A Binary Merge Square [5] is the adjacency matrix of a quotient graph. Consequently, if a strategy can operate on the adjacency matrix of the original graph, then the same strategy can cooperate with an merged adjacency matrix with an intermediate Merge Square as well. It introduces a dynamic reconsideration process where previous decisions of a strategy can be revised after each Merge Operation by exploiting the additional information contained in the intermediate matrices. *The author* in [97] showed the efficiency of such an extension.

The author in [96] applied the structural properties of the Merge Table Models in the meta-heuristics design. *The author* introduced a better granular fitness function than the traditional one for the evolutionary solvers of the colouring problem. This resulted in a smoother landscape of the objective function, which increased the efficiency of the optimisation process. Moreover *the author* defined a mutation which forces the difficult vertices by a Merge Table Model (for which the colouring is problematic) in advance in the merge/colour assignment.

Merge Algorithms

The author in [94; 96–102] combined various novel Merge Strategies with different Merge Frameworks and analysed their performance. The algorithms were compared with standard benchmark algorithms on various benchmark graphs. The experimental analysis showed that the novel Merge Algorithms perform well in the comparison. They generally outperformed the benchmark algorithms especially in the phase transition region where the problems become hard.

Conclusions

The new colouring approach presented in this thesis demonstrates that graph colouring can be effectively modelled by quotient or power graphs. It provides a potential reduction in computational cost, as well as a uniform and compact way in which algorithms can be defined. Embedding algorithms in the framework supports both their structural and performance comparison in a common way, which can be anyway problematic. The framework itself generalises a formal colouring approach. Due to this generalisation such an embedding an algorithm can be enhanced, resulting in new algorithms. The novel problem description results in novel information that can help us to extract and support a new scheme of the colouring process.

[5] Usually this extension can be applied on the other Merge Models as well.

Összefoglalás

Jelen értekezés összefoglalja a szerző elmúlt évekbeli munkásságát a gráf színezés területén. A szerző kifejlesztett egy általános keretrendszert gráf színezési algoritmusok számára, ahol a hagyományos színezés speciális gráf homomorfizmusokon keresztül került definiálásra, Zykov munkássága nyomán [161; 162]. Ezen homomorfizmusok hasznosnak bizonyultak az algoritmus tervezésben (lásd Juhos et al. ([94; 96–102]). Ezen összefoglaló az értekezés struktúráját követi.

Kvóciens és Hatvány Módszer

A szerző a gráfszínezési folyamatot kvóciens és hatványgráfok segítségével, gráf homomorfizmusokon sorozatával definiálta (lásd Juhos et al. [96; 100]). A homomorfizmusok az azonos színű csúcsok következetes összehúzásából vagy csoportba foglalásából származnak. A szerző Kvóciens és Hatvány Módszernek nevezte el az ezen elven alapuló színezési módszereit. Ezeknek célja egy olyan homomorfizmus megtalálása amely az eredeti gráfot egy megfelelő teljes gráfba vagy azzal homomorf gráfba képezi. Az így kapott homomorfizmus meghatároz egy színezést az eredeti gráfra. A szekvenciális színezési eljárások támogatása végett a tekintett homomorfizmus további homomorfizmusok egymásutánjaként, kompozíciójaként kerül előállításra, megadva egy ún. közbenső homomorfizmus sorozatot. Ezen homomorfizmusok hasznos közbenső gráf struktúrákat hoznak létre, amelyek vizsgálata hatékony színezési eljárásokat eredményeztek valamint a színezési folyamatba egy alternatív betekintést nyújtanak (lásd Juhos et al. [96–102]).

Merge Modellek

Gráf homomorfizmusok definiálják a kapcsolatot az eredeti és a kvóciens vagy hatvány gráf/multigráf között. A szerző definiált négy mátrix műveletet, amelyeket Merge Műveleteknek, vagy röviden Merge-nek nevezett el (lásd Juhos et al. [96; 100]). Egy Merge Művelet az eredeti gráf szomszédsági mátrixát képezi le egy mátrixba amely egy kvóciens gráf/multigráfot vagy hatvány gráf/multigráfot határoz meg, ezeket a szerző Bináris/Integer Merge Square-nek és Bináris/Integer Merge Table-nek, vagy összefoglaló nevükön Merge Mátrixoknak nevezte el. Egymást követő Merge Műveletek sorozata hoz létre egy hagyományos értelemben vett színezést. A Merge Mátrixok sorai független csúcs halmazokat határoznak meg. A színosztályok, valamint a csúcsok önmagukban is független csúcshalmazokat alkotnak. A Merge Műveletek hagyományos értelemben

ezek unióját jelentik. Ezen modelljét a színezésnek a szerző Merge Modellnek nevezte el. A modell támogatja a párhuzamos szoftver és hardver implementációt. Egy szekvenciális színezési algoritmus amely ezen modellre épül jelentős teljesítménybeli javulást könyvelhet el. A szerző ezen javulást elméletileg és tapasztalatilag is alátámasztotta (lásd Juhos et al. [99]) valamint hatékony új színezési eljárásokat dolgozott ki ezen modellek segítségével [94; 97–99; 101; 102].

Merge Keretrendszer

A Merge Modellek a gráfszínezést mátrix reprezentáció és speciális műveletek útján definiálják. A szerző kidolgozott három általános keretrendszert amelyek absztrakt színezési algoritmusokat határoznak meg (lásd Juhos et al. [100; 101]). Ezen absztrakciók az általánosításai a tradicionális színezési sémáknak. A Merge Műveletek helyettesítik a hagyományos értelemben vett színezést. A Merge Modellekben eltűnik a különbség a szín és a csúcs kiválasztási stratégiák között. Elegendő egy általános sorválasztási stratégiát meghatározni, amely alkalmas színezett vagy színezetlent sorok kiválasztására is, ha a tradicionális színezési sémákat akarjuk követni. Azonban itt a színek csak jelzés értékűek, jelzik, hogy egy sor érintett volt-e már a Merge Műveletben. Attól függően, hogy milyen sorrendben választjuk ki a különböző állapotú (színezett/színezetlen) sorokat kaphatunk két eltérő keretrendszert: vagy először egy színezetlen (Uncoloured) sort választunk, majd egy színezettet (Coloured) a Merge Művelethez (UC Merge Keretrendszer) vagy fordítva (CU Merge Keretrendszer). Ezen keretrendszerek általánosításai a hagyományos színezési sémának (lásd Juhos et al. [100]). Valójában nem szükséges megkülönböztetni a színezett és színezetlen státuszokat, egy kiválasztási stratégia választhatna tetszőleges két sort egy Merge Mátrixból, hogy végrehajtsa rajtuk a Merge Műveletet. Ez a megközelítés a CC Merge Keretrendszerben lett definiálva (lásd Juhos et al. [96]). Egy sor a Merge Mátrixban egy színosztályt azonosít, azaz független csúcshalmazt. Hagyományos értelemben a CC Merge Keretrendszerben egy algoritmus két színosztályt/független csúcshalmazt választ majd ezek unióját képezi. Ezen keretrendszerek az új színezési modellel támogatják az egységes algoritmus analízist (lásd Juhos et al. [97; 99; 101; 102]). Mindhárom keretrendszer egy egységes szerkezetet tükröz. Az algoritmusok ezen keretrendszerekben Merge Mátrix sorok sorozatos kiválasztását végzik, majd végrehajtanak rajtuk egy Merge Műveletet, mely eredményeképpen előáll egy színezés. Egyik általános keretrendszernek sincs konkrét sorkiválasztási stratégiája. A keretrendszerek konkrét kiválasztási stratégiákkal alkotnak algoritmusokat.

Merge Stratégiák

A Merge Algoritmusok minden lépésben egy Merge Műveleteket hajtanak végre a Merge Mátrix két kiválasztott során. A sorok kiválasztásához valamilyen kiválasztási stratégiára (stratégiákra) van szükség. A sorozatos Merge Műveletek végén az záró Merge Mátrix áll amelyen további Merge Művelet nem végezhető. A sorok kiválasztása a műve-

let végrehajtások során fontos ez határozza meg a színezés minőségét, azaz, hogy hány színt használtunk fel az elért színezésben. A szerző különböző sorkiválasztási stratégiákat, Merge Stratégiákat határozott meg, amelyek segítik a hatékony sorkiválasztást, melyeket elméletileg és tapasztalati úton is elemzett (lásd Juhos et al. [94; 96–102]). A felsorolt stratégiák Bináris Merge Modelleket feltételeznek, bár Integer Modellbeli párjuk is megadható. Az Integer Modellek az algoritmus tervezésben nyújtanak számos támogatást, mint például a visszalépés vagy a másodlagos döntéshozatal (lásd Juhos et al. [99]).

A leghosszabb Merge sorozat. Mivel a Merge Mátrix sorok színosztályokat azonosítanak. Ennélfogva a cél a sorok számának csökkentése. Ezt a leghosszabb Merge sorozattal létrehozásával érhetjük el. Ennek érdekében a szerző bevezetett két stratégiát (lásd Juhos et al. [97]). A Dot Product Stratégia nem-zéró elemek alakulását követi nyomon a Merge-k során. Megkísérli azok számát minimálisan tartani. A Bár a nem-zéró elemek meggátolhatják a Merge Műveleteket, a zéró elemek segítenek támogatják azokat. Így a Cosine Stratégia figyelembe veszi mindkettő alakulását a Merge-k során és annak megfelelően alakítja a sor kiválasztásokat.

Párhuzamos sorok. A Cosine Stratégia előnyben részesíti a párhuzamos sorokat a Merge Mátrixokban. Ez ésszerű választás azért is mert a szomszédsági mátrix sorai amelyek azonos színosztályhoz tartoznak egy optimális színezésben majdnem párhuzamosak. A Merge-k során a keletkező kvóciens gráfokhoz tartozó Merge Square Mátrixokban ez a párhuzamos tulajdonság egyre karakteresebbé válik. A Merge Square-ek ésszerű módosításai Karger et al. [103] munkássága nyomán további támogatást nyújt a Cosine stratégia számára. Felhasználva ezt az szerző definiálta a Zykov-fa és Lovász-theta stratégiát (lásd Juhos et. al. [94; 102])

Szín hasonlóság Valójában a Zykov-fa és Lovász-theta stratégia a csúcsok szín hasonlóságának becslésén alapszik. A szomszédsági mátrix egy szín különbözőségi relációt határoz meg, mivel a csúcsok amelyek (él-)relációban vannak nem színezhetők azonosan. Ennek ellenkezője a színezési reláció. Egy színezés megadható egy szín hasonlósági reláció meghatározásával, itt csak az azonos csúcsok állnak relációban. A reláció egy $\{0,1\}-$mátrixszal kifejezhető, ez a színezési mátrix. Ez megadja, hogy két csúcs azonos színű vagy különböző. Bár az optimális színezések mátrixa is megadható eképpen, ezek alkotják a feladat megoldását, tehát ezekre nem támaszkodhatunk. Noha ezek nem ismertek, az átlaguk közelíthető egy szemi-definit program megoldásával amely a Lovász-theta-t is szolgáltatja eredményül. Így egy közelített szín hasonlósági relációt kapunk. Amely egy valós értékű mátrixszal írható le, melyben a legnagyobb és legkisebb elemek fontos információt hordoznak. A szerző ezen információkat valamint Zykov munkásságát felhasználva (lásd [161; 162]) elkészítette Zykov-fa és Lovász-theta stratégiát. Ahol egy kvóciens gráf csúcsai összekötendők vagy Merge-lendők a kicsi illetve nagy közelített hasonlósági értékeknek megfelelően. Az összekötési (él hozzáadási) és Merge Műveletek során a hasonlóság egyre karakterisztikusabbá válik, támogatva ezzel az egyre értékesebb sor kiválasztásokat.

Norma minimalizálás az eredményben. A Dot Product Stratégia azt a két sort választja ki Merge-elésre, amelyeknek maximális a skaláris szorzatuk. Ez az eredmény

Merge Mátrixban az elemenkénti normák minimalizálását eredményezi. A záró Merge Mátrix, ami egy optimális megoldáshoz tartozik, rendelkezik a legkisebb elemenkénti mátrixnormával az összes lehetséges záró mátrix közül. Emiatt az elemenkénti norma minimalizálása ésszerű stratégia. Továbbá egy optimális zárómátrixnál a származtatott mátrixnormák is minimálisak. Ez a megfigyelés vezetett a szerző legnagyobb norma csökkentés stratégiájához (lásd Juhos et al. [101]). Speciális esetben ez a spektrálnorma minimalizálási stratégiához vezet, amely a legkisebb a származtatott normák között és ennélfogva jó karakterizációja egy mátrixnak. A szerző a spektrálnorma minimalizálási stratégiát elemezte, amely hatékonynak bizonyult az elemzések során. A spektrálnorma próba Merge-ket kell, hogy végezzen. Az eredmény mátrix normája határozza meg a kiválasztási stratégiát. Ez számításigényes feladat. Merikoski és Kumar megadott több hatékony spektrálnorma közelítési formulát. (lásd [123]). Felhasználva ezen formulákat a szerző adaptálta a Spektrálnorma stratégiát és közelített spektrálnorma stratégiákat vezetett be (lásd Juhos et al. [101]). A közelítéssel lehetőség nyílik a választási stratégia közvetlen meghatározására az aktuális Merge Mátrixból próba Merge-ek nélkül. Továbbá a közelítő formula rámutat a Dot Product Stratégiával való hasonlóságra.

Mátrix tulajdonságok – Merge Útvonal A Merge sorozatok nyomán a mátrixok tulajdonságai követhetők. A kívánt mátrix tulajdonságokból alkossunk egy tulajdonságvektort. Ezek a vektorok alkothatják az alapját a kiválasztási stratégiáknak. Az egymást követő tulajdonságvektorok egy útvonalat, a Merge útvonalat, határoznak meg (lásd Juhos et al. [101]). Ez az útvonal elemei összefüggésben vannak a színezés lépéseivel az útvonal vége pedig a színezés jóságával. Az ideális útvonal amely optimális színezéshez vezethet nem ismert, mert a feladat egy ilyen útvonal megtalálása. Az optimális útvonal kezdő és a végpontjai általában ismertek, a szerző bevezetett egy általános stratégiát amely az optimális Merge Útvonal közelítésén alapszik (lásd Juhos et al. [101]) felhasználva egy előzetes tudást. Az előzetes tudás megszerzése a szerző a Merge Útvonal koncepciót intelligens tanulási és klaszterezési eljárásokkal ötvözte (lásd Juhos et al. [95]).

Kiterjesztett heurisztikák és meta-heurisztikák A szerző a nem Merge alapú színezési stratégiák egy Merge kiterjesztését határozta meg, az illető stratégiák egy megfelelő Merge Modellbe való beágyazásával (lásd Juhos et al. [97]). A kiterjesztet stratégiák teljesítményének elméleti és tapasztalati vizsgálata javulást mutatott az eredetihez képest. A kiterjesztés egyik a Bináris Merge Square-ek példáján egyszerűen nyomon követhető, habár általában a kiterjesztés a többi Merge Modellre is érvényes. A szomszédsági mátrixa egy kvóciens gráfnak. Így egy stratégia amely az eredeti gráf szomszédsági mátrixán működik, az egy Merge Square Modellel képes együttműködni. Ez lehetőséget biztosít egy dinamikus felülvizsgálati eljárásra amely során minden Merge Művelet után, a stratégia képes előző döntéseit megváltoztatni, azon új információk alapján amely a keletkező Merge Mátrixban elérhető.

A Merge Modellek strukturális jellemzői támogatást nyújtanak meta-heurisztikák tervezéséhez is (lásd Juhos et al. [96]). A szerző gráfszínezési evolúciós algoritmusok számára definiált a Merge Table modellek segítségével egy finomított fitnesz függvényt

a hagyományosan alkalmazott fitnesz javításaként. Melynek eredményeként egy simább optimalizálási felületet kapunk, amely növeli az optimalizálás hatékonyságát. Továbbá egy mutáció operátort definiált amely a színezésben a Merge Modellek alapján nehezen színezhető csúcsokat előreveszi a színezési folyamatban.

Merge Algoritmusok

A szerző kombinálta a Merge Stratégiáit a különböző Merge Keretrendszereivel valamint elemezte ezek teljesítményét (lásd Juhos et al. [94; 96–102]). Az így keletkezett algoritmusok összehasonlításra kerültek standard 'benchmark' eljárásokkal számos 'benchmark' gráfon. A kísérleti eredmények igazolták a szerző algoritmusainak hatékonyságát, melyek általában felülmúlták a 'benchmark' eljárásokat különösképpen az ún. 'phase transition' területen ahol az igazán nehéz problémák találhatók.

Konklúzió

A szerző új színezési megközelítése a gráfszínezés hatékony modelljének bizonyult. Jelentős csökkentést hozhat az algoritmusok számítási komplexitásában. Továbbá egységes és tömör leírását biztosítja a színezési eljárásoknak, biztosítva ezzel az egységes szerkezetben vett strukturális elemzését. Az algoritmusok implementálása ezen közös módon lehetőséget biztosít az egységes teljesítmény mérésre. Az új színezési keretrendszer általánosítja az eddig színezési sémákat. Ezen általánosítás következményeként egy algoritmus beágyazása a modellbe annak kibővítése mellett teljesítmény javulással is járhat. Ezen új megközelítés új információ kinyerési technikákat is támogat amely az algoritmus tervezésben segíthet valamint új irányokat adhat a probléma elemzéséhez.

Bibliography

[1] Aardal KI, van Hoesel SPM, Koster AMCA, Mannino C, Sassano A: **Models and solution techniques for frequency assignment problems**. *Annals of Operations Research* 2007, **153**:79–129.

[2] Alon N: **Graph Powers, in: Contemporary Combinatorics**. *Bolyai Society Mathematical Studies* 2002, **4**:11–28.

[3] Andrews GR: *Foundations of Multithreaded, Parallel, and Distributed Programming*. Addison-Wesley 2000.

[4] Appel K, Haken W: **Solution of the Four Color Map Problem**. *Scientific America* 1977, **237**:108–121.

[5] Avanthay C, Hertz A, Zufferey N: **A variable neighborhood search for graph coloring**. *European Journal of Operational Research* 2003, **151**:379–388.

[6] Bäck T: *Evolutionary Algorithms in Theory and Practice*. Oxford University Press, New York 1996.

[7] Back T, Fogel DB, Michalewicz Z (Eds): *Handbook of Evolutionary Computation*. Bristol, UK, UK: IOP Publishing Ltd. 1997.

[8] Beck A, Bleicher MN, Crowe DW: *Excursions into Mathematics: The Millennium Edition*. A K Peters, Ltd. 2000.

[9] Biggs N: *Algebraic graph theory*. Cambridge University Press 1994.

[10] Blochliger I, Zufferey N: **A graph coloring heuristic using partial solutions and a reactive tabu scheme.** *Computers & Operations Research* 2008, **35**:960–975.

[11] Bollobás B, Borgs C, Chayes J, Kim J, Wilson D: **The scaling window of the 2-SAT transition**. *Random Structures and Algorithms* 2001, **18**(3):201–256.

[12] Bollobás B, Erdős P: **Cliques in random graphs**. *Mathematical Proceedings of the Cambridge Philosophical Society* 1976, **80**:419–427.

[13] Bollobás B: **The chromatic number of random graphs**. *Combinatorica* 1988, **8**:49–55.

[14] Bollobás B: *Modern Graph Theory*. Springer 1998.

[15] Bollobás B, Catlin PA, Erdős P: **Hadwiger's conjecture is true for almost every graph**. *European Journal on Combinatorics* 1980, **1**:195–199.

[16] Bondy JA, Murty USR: *Graph Theory with Applications*. North-Holland 1976.

[17] Brèlaz D: **New methods to color the vertices of a graph**. *Communications of the ACM* 1979, **22**:251–256.

[18] Briggs P, Cooper K, Kennedy K, Torczon L: **Coloring heuristics for register allocation**. *In ASCM Conference on Program Language Design and Implementation* 1989, :275–284.

[19] Briggs P, Cooper K, Torczon L: **Improvements to graph coloring register allocation**. *ACM Transactions on Programming Languages and Systems* 1994, **16**(3):428–455.

[20] Brooks RL: **On colouring the nodes of a network**. *Proceedings of the Cambridge Philosophical Society, Math. Phys. Sci.* 1941, **37**:194–197.

[21] Bui TN, Patel CM: **An ant system algorithm for coloring graphs**. In *Proceedings of the Computational Symposium on Graph Coloring and its Generalizations* 2002:83–91.

[22] Campelo M, Correa R, Frota Y: **Cliques, holes and the vertex coloring polytope**. *Information Processing Letters* 2004, **89**:159–164.

[23] Caramia M, Dell'Olmo P: **Bounding vertex coloring by truncated multistage branch and bound**. *Networks* 2004, **44**:231–242.

[24] Caramia M, Dell'Olmo P: **Coloring graphs by iterated local search traversing feasible and infeasible solutions**. *Discrete Applied Mathematics* 2008, **156**:201–217.

[25] Chaitin G: **Register allocation and spilling via graph coloring**. *SIGPLAN Not.* 2004, **39**:66–74.

[26] Chaitin G: **Register allocation and spilling via graph coloring**. In *ACM SIGPLAN 82 Symposium on Compiler Construction*, ACM Press 1982:98–105.

[27] Chaitin G, Auslander M, Chandra A, Cocke J, Hopkins M, Markstein P: **Register allocation via coloring.** *Computer Languages* 1981, **6**:47–57.

[28] Chams M, Hertz A, de Werra D: **Some experiments with simulated annealing for coloring graphs**. *European Journal of Operational Research* 1987, **32**:260–266.

[29] Cheeseman P, Kanefsky B, Taylor WM: **Where the Really Hard Problems Are**. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence, IJCAI-91, Sidney, Australia* 1991:331–337.

[30] Chiarandini M, Dumitrescu I, Stützle T: **Very large-scale neighborhood search: Overview and case studies on coloring problems**. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence, IJCAI-91, Sidney, Australia* 2008.

[31] Chow FC, Hennessy JL: **The priority-based coloring approach to register allocation**. *ACM Transactions on Programming Languages and Systems* 1990, **12**:501–536.

[32] CNET: **Xbox specs revealed** 2005, [http://cnet.com/Xbox+specs+revealed/2100-1043_3-5705372.html]. [Accessed: Nov. 10, 2005].

[33] Coll P, Marenco J, Méndez Díaz I, P Z: **Facets of the graph coloring polytope**. *Annals of Operations Research* 2002, **116**:79–90.

[34] Comba J, Dietrich C, Pagot C, Scheidegger C: **Computation on GPUs: from a programmable pipeline to an efficient stream processor**. *Revista de Informatica Teorica e Aplicada* 2003, **10**:41–70.

[35] Cook SA: **The Complexity of Theorem-Proving Procedures**. ACM Press 1971:151–158.

[36] Cooper KD, Dasgupta A: **Tailoring Graph-coloring Register Allocation For Runtime Compilation**. In *2006 International Symposium on Code Generation and Optimization (CGO'06)* 2006.

[37] Corneil DG, Graham D: **An algorithm for the chromatic number of a graph**. *SIAM Journal of Computing* 1973, **2**:311–318.

[38] Corporation N: **NVIDIA tesla: GPU computing technical brief, version 1.0.0.** *http://www.nvidia.co.uk/content/PDF/Tesla product literature/tesla technical brief.pdf* 2007.

[39] Costa D, Hertz A: **Ants Can Colour Graphs**. *Journal of the Operations Research Society* 1997, **48**:295–305.

[40] Costa D, Hertz A, Dubuis C: **Embedding a sequential procedure within an evolutionary algorithm for coloring problems in graphs.** *Journal of Heuristics* 1995, **1**:105–128.

[41] Coudert O: **Exact coloring of real-life graphs is easy**. In *DAC '97: Proceedings of the 34th annual conference on Design automation*, New York, NY, USA: ACM 1997:121–126.

[42] Craenen B, Eiben A, van Hemert J: **Comparing Evolutionary Algorithms on Binary Constraint Satisfaction Problems**. *IEEE Transactions on Evolutionary Computation* 2003, **7**(5):424–444.

[43] Culberson J, Gent I: **Frozen development in graph coloring**. *Theor. Comput. Sci.* 2001, **265**(1–2):227–264.

[44] Culberson J: **Iterated Greedy Graph Coloring and the Difficulty Landscape**. Tech. Rep. TR 92-07 1992.

[45] Cvetković D, Rowlinson P: **The largest eigenvalue of a graph - a survey**. *Linear and Multilinear Algebra* 1990, **28**:3–33.

[46] Cvetković DM, Doob M, Sachs H: *Spectra of Graphs: Theory and Applications*. Academic Press 1980.

[47] de Werra D: **An introduction to timetabling.** *European Journal of Operations Research* 1985, **19**:151–162.

[48] de Werra D: **Heuristics for Graph Coloring**. *Computational Graph Theory* 1990, **Comput. Suppl. 7**:191–208.

[49] Diestel R: *Graph theory*. Springer 2000.

[50] Dukanovic I, Rendl F: **Semidefinite programming relaxations for graph coloring and maximal clique problems**. *Mathematical Programming, Serie B* 2007, **109**:345–365.

[51] Dukanovic I, Rendl F: **A semidefinite programming-based heuristic for graph coloring**. *Discrete Applied Mathematics* 2008, **156**(2):180–189.

[52] Eiben AE, van Der Hauw JK, van Hemert JI: **Graph Coloring with Adaptive Evolutionary Algorithms**. *Journal of Heuristics* 1998, **4**:25–46.

[53] Eiben A, Smith J: *Introduction to Evolutionary Computing*. Springer 2007.

[54] Erdős P: **On cliques in graphs**. *Israel Journal of Mathematics* 1966, **4**:233–234.

[55] Falkenauer E: *Genetic Algorithms and Grouping Problems*. John Wiley 1998.

[56] Feige U, Langberg M, Schechtman G: **Graphs with Tiny Vector Chromatic Numbers and Huge Chromatic Numbers.** In *Proceedings of the 43rd Symposium on Foundations of Computer Science*, IEEE Computer Society 2002:283–292.

[57] Figueiredo R, Barbosa V, Maculan N, de Souza C: **New 0 – 1 integer formulations of the graph coloring problem**. In *Proceedings of XI CLAIO* 2002.

[58] Frieze AM, Krivelevich M, Smyth CD: **On the Chromatic Number of Random Graphs with a Fixed Degree Sequence**. *Combinatorics, Probability & Computing* 2007, **16**(5):733–746.

[59] Galinier P, Hao JK: **Hybrid evolutionary algorithms for graph coloring**. *Journal of Combinatorial Optimization* 1999, **3**:379–397.

[60] Galinier P, Hertz A: **A survey of local search methods for graph coloring**. *Computers & Operations Research* 2006, **33**:2547–2562.

[61] Gamst A: **Some lower bounds for a class of frequency assignment problems**. *IEEE Transactions of Vehicular Echnology* 1986, **35**:8–14.

[62] Garey MR, Johnson DS: *Computers and Intractability: A Guide to the Theory of NP-Completeness*. New York: W.H. Freeman & Company 1979.

[63] George L, Appel A: **Iterated register coalescing**. *ACM Transactions on Programming Languages and Systems* 1996, **18**(3):300–324.

[64] Gerschgorin S: **Über die Abgrenzung der Eigenwerte einer Matrix**. *Akad. Nauk. USSR Otd. Fiz.-Mat. Nauk.* 1931, **7**:749–754.

[65] Gethner E, Springer: **How False Is Kempe's Proof of the Four-Color Theorem?** *Congr. Numer.* 2003, **164**:159–175.

[66] Gibbons LE, Hearn DW, Pardalos PM, Ramana MV: **Continuous Characterizations of the Maximum Clique Problem**. *Mathematics of Operations Research* 1996, **22**:754–768.

[67] Gintaras P: **On the Graph Coloring Polytope**. *Information technology and control* 2008, **37**:7–11.

[68] Glass CA, Prügel-Bennett A: **Genetic algorithms for graph colouring: Exploration of Galinier and Hao's algorithm.** *Journal of Combinatorial Optimization* 2003, **7**:229–236.

[69] Golub GH, van Loan CF: *Matrix Computations*. Hopkins Fulfillment Service 1996.

[70] Graham R, Grötschel M, Lovász L (Eds): *Handbook of Combinatorics*. North-Holland 2005.

[71] Grimmet G, McDiarmid C: **On colouring random graphs**. *Mathematical Proceedings of the Cambridge Philosophical Society* 1975, **77**:313–324.

[72] Grötschel M, Lovász L, Schrijver A: *Geometric Algorithms and Combinatorial Optimization*. Springer 1988.

[73] Guruswami V, Khanna S: **On the hardness of 4-coloring a 3-colorable graph**. In *Proceedings of the 15th Annual IEEE Conference on Computational Complexity* 2000:188–197.

[74] Hadwiger H: **Über eine Klassifikation der Streckenkomplexe**. *Vierteljschr. Naturforsch. ges. Zürich* 1943, **88**:133–143.

[75] Hajnal P: **Eigenvalues of graphs: Graph colouring and the Perron-Frobenius eigenvector**. *Péter Hajnal's Combinatorics seminars. University of Szeged.* 2004.

[76] Hajös G: **Über eine Konstruktion nicht n-färbbarer Graphen**. *Wiss.Z.Martin-Luther-Univ.Halle-Wittenberg Math.-Naturw. Reihe* 1961, **10**:116–117.

[77] Halldórsson MM: **A Still Better Performance Guarantee for Approximate Graph Coloring**. *Inf. Process. Lett.* 1993, **45**:19–23.

[78] Halmos PR: *Finite-Dimensional Vector Spaces*. Springer 1974.

[79] Hamiez JP, Hao JK: **Scatter search for graph coloring**. In *Artificial Evolution, Volume 2310* 2001:168–179.

[80] Harary F: *Graph Theory*. Reading, MA: Addison-Wesley 1994.

[81] Harmanani H, Abas H: **A method for the minimum coloring problem using genetic algorithms**. In *MS'06: Proceedings of the 17th IASTED international conference on Modelling and simulation*, Anaheim, CA, USA: ACTA Press 2006:487–492.

[82] Hastad J: **Clique is hard to approximate within** $n^{(1-\epsilon)}$. *Acta Mathematica* 1999, **182**:105–142.

[83] Heawood P: **Map-colour theorem**. *Quarterly Journal of Mathematics* 1890, **24**:332–338.

[84] Heawood P: **On the four-colour map theorem**. *Quarterly Journal of Mathematics* 1898, **29**:270–285.

[85] Hell P, Nešetřil J: **On the complexity of H-coloring**. *Journal of Combinatorial Theory* 1990, **48**:92–110.

[86] Hell P, Nešetřil J: *Graphs and Homomorphisms*. Oxford Lecture Series in Mathematics and Its Applications, Oxford University Press 2004.

[87] Hertz A, de Werra D: **Using tabu search techniques for graph coloring**. *Computing* 1987, **39**:345–351.

[88] Hoffman AJ: **On eigenvalues and colorings of graphs**. In *Graph Theory and its Applications*. Edited by Harris B, Academic Press 1970:79–91.

[89] Horn RA, Johnson CR: *Norms for Vectors and Matrices*. Cambridge University Press 1990.

[90] Jensen TR, Toft B: *Graph coloring problems*. Wiley-Interscience 1995.

[91] Johnson DS, Mehrotra A, Trick MA: **Special issue on computational methods for graph coloring and its generalizations**. *Discrete Applied Mathematics* 2008, **156**:145–146.

[92] Johnson D, Aragon C, McGeoch L, Schevon C: **Optimization by simulated annealing: an experimental evaluation; part II, graph coloring and number partitioning**. *Operations Research* 1991, **39**:378–406.

[93] Johnson D, Trick M: *Cliques, Coloring, and Satisfiability*. American Mathematical Society, DIMACS 1996.

[94] Juhos I: **Graph Colouring through Clustering**. *Seminar. University of Edinburgh.* 2009.

[95] Juhos I, Szarvas G: **Intelligent Forecast with Dimension Reduction**. In *Applied Soft Computing Technologies: The Challenge of Complexity, Volume 34 of* Advances in Soft Computing. Edited by Abraham A, de Baets B, Köppen M, Nickolay B, Springer 2006:279–292.

[96] Juhos I, Tóth A, van Hemert J: **Binary Merge Model Representation of the Graph Colouring Problem**. In *Evolutionary Computation in Combinatorial Optimization, Volume 3004 of* Lecture Notes in Computer Science. Edited by Gottlieb J, Raidl, Raidl GR, Springer 2004:124–134.

[97] Juhos I, Tóth A, van Hemert J: **Heuristic Colour Assignment Strategies for Merge Models in Graph Colouring**. In *Evolutionary Computation in Combinatorial Optimization, Volume 3448 of* Lecture Notes in Computer Science. Edited by Gottlieb J, Raidl, Raidl GR, Springer 2005:132–143.

[98] Juhos I, van Hemert J: **Improving graph colouring algorithms and heuristics using a novel representation**. In *Evolutionary Computation in Combinatorial Optimization, Volume 3906 of* Lecture Notes in Computer Science. Edited by Gottlieb J, Raidl, Raidl GR, Springer 2006:123–134.

[99] Juhos I, van Hemert J: **Increasing the efficiency of graph colouring algorithms with a representation based on vector operations.** *Journal of Software* 2006, **1**(2):24–33.

[100] Juhos I, van Hemert J: *Contraction-based heuristics to improve the efficiency of algorithms solving the graph colouring problem*, Springer, *Volume 153 of* Studies in Computational Intelligence 2008 chap. III, :175–192.

[101] Juhos I, van Hemert J: **Graph Colouring Heuristics Guided by Higher Order Graph Properties**. In *Evolutionary Computation in Combinatorial Optimization, Volume 4972 of* Lecture Notes in Computer Science. Edited by van Hemert J, Cotta C, Springer 2008:97–109.

[102] Juhos I, van Hemert JI: **Graph colouring through clustering by Zykov-tree and Lovász-theta**. *Submitted* 2009.

[103] Karger D, Motwani R, Sudan M: **Approximate graph coloring by semidefinite programming**. *Journal of the ACM* 1998, **45**(2):246–265.

[104] Karp RM: *Complexity of Computer Computations: Reducibility Among Combinatorial Problems*, New York: Plenum 1972 :85–103.

[105] Knuth DE: **The Sandwich Theorem**. *Electronic J. Combinatorics* 1994, **1**:1.

[106] Kubale M: *Graph Colorings*. American Mathematical Society 2004.

[107] Larsen M, Propp J, Ullman D: **The Fractional Chromatic Number Of Mycielski's Graphs**. *J. Graph Theory* 1995, **19**:411–416.

[108] Leighton F: **A graph coloring algorithm for large scheduling problems**. *Journal of Reasearch of the National Bureau of Standards* 1979, **84**:489–506.

[109] Lovász L: **Normal hypergraphs and the weak perfect graph conjencture**. *Discrete Mathematics* 1972, **2**:253–267.

[110] Lovász L: **On the Shannon capacity of a graph**. *IEEE Transaction Information Theory* 1979, **25**:1–7.

[111] Lovász L: **Perfect graphs**. In *More Selected Topics in Graph Theory*. Edited by L W Beineke RLW, Academic Press 1983:55–57.

[112] Lovász L: *Combinatorial Problems and Exercises*, North Holland 1993 :69–69.

[113] Lueh GY, Gross T, Adl-Tabatabai AR: **Global Register Allocation Based on Graph Fusion**. In *Languages and Compilers for Parallel Computing* 1996:246–265.

[114] Malaguti E, Toth P: **A survey on vertex coloring problems**. *International Transactions in Operational Research* 2009, :1–34.

[115] Marino A, Damper R: **Breaking the Symmetry of the Graph Colouring Problem with Genetic Algorithms**. In *Late Breaking Papers at the 2000 Genetic and Evolutionary Computation Conference*. Edited by Whitley D 2000:240–245.

[116] Matula D, Marble G, J I: *Graph coloring algorithms*, Academic Press 1972 :109–122.

[117] McDiarmid C: **Coloring random graphs badly**. *Reso Notes Mathematical* 1979, (34):76–86.

[118] Mehrotra A, Trick MA: **A column generation approach for graph coloring**. INFORMS *Journal on Computing* 1996, **8**:344–354.

[119] Mehta DP, Sahni S: *Handbook Of Data Structures And Applications (Chapman & Hall/Crc Computer and Information Science Series.)*, Chapman & Hall/CRC 2004 chap. 59.4.2.

[120] Méndez-Díaz I, Zabala P: **A polyhedral approach for graph coloring**. *Electronic Notes in Discrete Mathematics* 2001, **7**:178–181.

[121] Méndez-Díaz IP Zabala: **A cutting plane algorithm for graph coloring**. *Discrete Applied Mathematics* 2008, **156**:159–179.

[122] Méndez-Díaz I, Zabala P: **A branch-and-cut algorithm for graph coloring**. *Discrete Appl. Math.* 2006, **154**(5):826–847.

[123] Merikoski JK, Kumar R: **Lower Bounds for the Spectral Norm**. *Journal of Inequalities in Pure and Applied Mathematics* 2005, **6**(3).

[124] Meurdesoif P: **Strengthening the Lovász Theta(G) bound for graph coloring.** *Math. Program.* 2005, **102**(3):577–588.

[125] Mohar B, Thomassen C: *Graphs on Surfaces*. The Johns Hopkins University Press, Baltimore and London 2001.

[126] Monasson R, Zecchina R, Kirkpatrick S, Selman B, Troyansky L: **Determining computational complexity from characteristic phase transitions**. *Nature* 1999, **400**:133–137.

[127] Motzkin TS, Straus EG: **Maxima for graphs and a new proof of a theorem of Turán**. *Canadian Journal of Mathematics* 1965, **17**:533–540.

[128] Mycielski J: **Sur le coloriage des graphes (for colouring graphs)**. *Colloquim Mathematiques* 1955, **3**(161):161–162.

[129] Ogawa H: **Labeled point pattern matching by Delaunay triangulation and maximal cliques**. *Pattern Recognition* 1986, **19**:35–40.

[130] Opsut R, Roberts FS: **On the fleet maintenance, mobile radio frequency, task assignment and traffic phasing problems**. In *The Theory and Applications of Graph*. Edited by Chartrand G, Alavi Y, Goldsmith D, Lesniak-Foster L, Lick D, John Wiley & Sons. 1981:479–492.

[131] Palubeckis G: **On the recursive largest first algorithm for graph colouring**. *Int. J. Comput. Math.* 2008, **85**(2):191–200.

[132] Park J, Moon SM: **Optimistic register coalescing**. *ACM Transactions on Programming Languages and Systems* 2004, **26**(4):735–765.

[133] Porumbel D, Hao JK, Kuntz P: **Diversity control and multi-parent recombination for evolutionary graph coloring algorithms**. In *LNCS 5482*, Springer 2009.

[134] Povh J, Rendl F, A W: **A boundary point method to solve semidefinite programs**. *Computing* 2006, **78**:277–286.

[135] Schaerf A: **A Survey of Automated Timetabling**. *Artificial Intelligence Review* 2004, **13**:87–127.

[136] Schindl D: **Graph coloring and linear programming**. In *Artificial Intelligence Review*, Presentation at First Joint Operations Research Days, Ecole Polytechnique Fédérale de Lausanne (EPFL), Available on line 2003.

[137] Sewell E: **An improved algorithm for exact graph coloring**. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science* 2006, **26**:359–373.

[138] Skiena S: *Implementing Discrete Mathematics: Combinatorics and Graph Theory with Mathematica*. Reading, MA: Addison-Wesley 1990.

[139] Skiena SS: *The Algorithm Design Manual.*, Springer-Verlag 1997 chap. 6.2.3 and 8.5.1, :144 and 312–314.

[140] Strang G: *Linear Algebra and its Applications*. San Diego 1988.

[141] Thompson CJ, Hahn S, Oskin M: **Using modern graphics architectures for general-purpose computing: a framework and analysis**. In *in MICRO 35: Proceedings of the 35th annual ACM/IEEE international symposium on Microarchitecture*, Los Alamitos, CA, USA: IEEE: Computer Society Press 2002:306–317.

[142] Toh KC: **Solving large scale semidefinite programs via an iterative solver on the augmented systems**. *SIAM Journal on Optimization* 2004, **14**:670–698.

[143] Trick M: **Computational Series: Graph Coloring and its Generalizations** 2003. [Http://mat.gsia.cmu.edu/COLORING03].

[144] University SK, Khanna S, Linial N: **On the Hardness of Approximating the Chromatic Number** 1993.

[145] van Hemert J: **Application of Evolutionary Computation to Constraint Satisfaction and Data Mining**. *PhD thesis*, Leiden University 2002.

[146] Vassilakis C: **An Optimisation Scheme for Coalesce/Valid Time Selection Operator Sequences**. *SIGMOD Record* 2000, **29**:38–43.

[147] Vegdahl SR: **Using node merging to enhance graph coloring**. In *PLDI '99: Proceedings of the ACM SIGPLAN 1999 conference on Programming language design and implementation*, New York, NY, USA: ACM Press 1999:150–154.

[148] Vincze A: **Star chromatic number**. *Journal of Graph Theory* 1998, **12**:551–559.

[149] Virágh J: *Numerikus matematika*. JATE Press 1997.

[150] Weisstein EW: *CRC Concise Encyclopedia of Mathematics*. Chapman & Hall/CRC, 2nd edition 2002.

[151] Welsh DJA, Powell MB: **An upper bound for the chromatic number of a graph and its application to timetabling problems**. *The Computer Journal* 1967, **10**:85–86.

[152] Wigderson A: **Improving the performance for approximate graph coloring**. *Journal of the ACM* 1983, **30**:729–735.

[153] Wilf HS: **The eigenvalues of a graph and its chromatic number**. *Journal of London Math. Soc.* 1967, :330–332.

[154] Wilf HS: **Spectral bounds for the clique and independence numbers of graphs**. *Journal of Combinatorial Theory* 1986, **40**:113–117.

[155] Willard S: *General Topology*. Addison-Wesley 1970.

[156] Wilson RM, van Lint JH: *A Course in Combinatorics*. Cambridge University Press 2001.

[157] Wilson R: *Four Colors Suffice: How the Map Problem Was Solved*. Princeton University Press 2004.

[158] Woeginger G: **Exact Algorithms for NP-Hard Problems: A Survey**. In *Four Colors Suffice: How the Map Problem Was Solved* 2003:185–207.

[159] Zhu X: **Circular chromatic number**. *Discrete Mathematics* 2001, **229**:371–410.

[160] Zuckerman D: **Linear degree extractors and the inapproximability of Max Clique and Chromatic Number**. *Theory of Computing* 2007, **3**:103–128.

[161] Zykov AA: **On some properties of linear complexes (in Russian)**. *Math. Sbornik* 1949, **24**:163–188.

[162] Zykov AA: **On some properties of linear complexes**. *American Mathematical Society Translations* 1952, **79**:81.